Издательство «Знание»

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Новое в жизни, науке, технике

Подписная научно-популярная серия

Издается ежемесячно с 1988 г.

Язык персональных вычислений

1990

4

Новое
в жизни,
науке,
технике

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЕЁ ПРИМЕНЕНИЕ

Подписная
научно-
популярная
серия

4/1990

ЯЗЫК ПЕРСОНАЛЬНЫХ ВЫЧИСЛЕНИЙ

Издается
ежемесячно
с 1988 г.

В номере:

МИР ЯЗЫКА АПЛ

В. М. Куприянов
ЯЗЫКИ, ПРОГРАММИСТЫ И ПРОГРАММЫ

А. И. Мирошников
ИСТОРИЧЕСКИЙ ОЧЕРК

А. В. Кондрашев
ЧТО ТАКОЕ АПЛ?

О. П. Лукша
АПЛ В АНАЛИЗЕ ДАННЫХ

А. О. Скоморохов
ВВЕДЕНИЕ В ЭКСПЕРТНЫЕ СИСТЕМЫ НА АПЛ

РУБРИКИ
БК за рога
Переводы

Издательство
«Знание»
Москва
1990

ББК 32.97
Я 40

Авторы выпуска

КОНДРАШЕВ Андрей Владимирович — младший научный сотрудник ВЦ АН СССР, автор семейства отечественных АПЛ-систем для мини-, микро- и персональных ЭВМ.

КУПРИЯНОВ Вячеслав Михайлович — кандидат физико-математических наук, доцент, старший научный сотрудник Физико-энергетического института ГК ИАЭ СССР. Область научной деятельности — обработка данных экспериментальных исследований на ЭВМ.

ЛУКША Олег Павлович — старший преподаватель ЦИПК ГК ИАЭ СССР. Области научной и педагогической деятельности — моделирование технологических процессов, прикладная статистика, компьютерная технология анализа данных, автоматизация учебного процесса с использованием ЭВМ.

МИРОШНИКОВ Алексей Иванович — кандидат технических наук, старший научный сотрудник Северо-Западного филиала ЦЭНИИ при Госплане РСФСР. Занимается разработкой систем поддержки решений с использованием языка АПЛ.

СКОМОРОХОВ Александр Олегович — кандидат технических наук, старший научный сотрудник Физико-энергетического института ГК ИАЭ СССР. Специалист в области применения современных информационных технологий в задачах технической диагностики АЭС.

РЕДАКТОР Б. М. ВАСИЛЬЕВ

ISBN 5-07-001245-2

© Издательство «Знание», 1990 г.

МИР ЯЗЫКА АПЛ

По-видимому, ни один из языков программирования не вызывает вокруг себя столько споров и взаимоисключающих суждений, как АПЛ. Крупнейший теоретик программирования Дж. Мартин сказал, что «АПЛ — это язык программирования сверхвысокого уровня, единственный пока кандидат в языки пятого поколения». Другой не менее авторитетный ученый Э. Дейкстра как-то назвал АПЛ «ошибкой, доведенной до совершенства». Ф. Брукс в своей книге «Мифический человеко-месяц» заметил, что если бы ему пришлось заново переписывать операционную систему для IBM/360, он бы сначала смакетировал ее на АПЛ. В одной отечественной публикации некто «обозвал» АПЛ китайским БЕЙСИКом.

В СССР АПЛ практически неизвестен, и это несмотря на то, что, по некоторым оценкам, количество людей по всему миру, так или иначе использующих его в своей практической деятельности, исчисляется сотнями тысяч. Авторы, которые на протяжении вот уже ряда лет активно и с успехом работают с АПЛ, решили написать брошюру, которая сейчас перед вами, с тем чтобы и отечественные пользователи ЭВМ смогли открыть для себя и, как мы надеемся, полюбить этот удивительный мир языка АПЛ.

Полюбить — значит понять...

В. М. КУПРИЯНОВ

ЯЗЫКИ, ПРОГРАММИСТЫ И ПРОГРАММЫ

Как беден наш язык. Хочу и не могу!
Не передать того ни другу, ни врагу,
Что буйствует в душе прозрачною волною.

А. Фет

В настоящее время в мире придумано несколько сотен машинных языков широкого применения, и число их продолжает расти. Интересно, что наиболее используемыми оказались далеко не самые удобные и отвечающие каким-либо формальным критериям качества языки — ФОРТРАН, БЕЙСИК и КОБОЛ.

Причин этого «вавилонского столпотворения» в программировании несколько, но главная заключается в том, что всякий искусственный язык ограничен и способен решать лишь достаточно узкий круг задач из всего спектра человеческой деятельности. Как только вычислительная техника начинает применяться в какой-то новой сфере или как только к программам предъявляются новые требования по их качеству, так сразу же выясняется, что ни один из уже существующих языков не способен в полной мере удовлетворить эту новую потребность. Тогда создается новый язык. По-видимому, этот процесс бесконечен, как бесконечен процесс познания. Таким образом, большинство споров о том, какой язык лучше, а какой хуже, не имеет под собой достаточно серьезных оснований. Оппоненты всегда находят «убийственные», с их точки зрения, аргументы, игнорируя при этом возражения противной стороны. Можно с пеной у рта доказывать убогость конструкций ФОРТРАНа, но это никак не отразится на его популярности, скажем, в среде физиков-экспериментаторов, для которых его «бессмертие» гарантируется десятками томов «Сборников текстов программ на ФОРТРАНе», где почти наверняка отыщется нужный в данный момент вычислительный алгоритм.

Нет хороших или плохих языков программирования. Есть лишь языки, лучше или хуже приспособленные для решения данного круга задач, стоящих перед каждым конкретным пользователем ЭВМ. Выбор программистом языка должен обеспечить ему наиболее благоприятные условия для создания хороших программ в его предметной области.

Здесь мы подошли к весьма непростому вопросу: «А что такое хорошая программа?» Программисты-профессионалы обычно отвечают на этот вопрос так: «Хорошая программа — это такая, которая работает как можно быстрее и расходует как можно

меньше ресурсов ЭВМ». Этот типичный ответ на современном уровне развития мировой информационной индустрии в большинстве случаев оказывается неверным. В те времена, когда вычислительная техника была очень дорогой, а ее возможности очень ограниченными, в основу критериев качества программирования действительно закладывались соотношения экономии аппаратных ресурсов. Именно тогда появились языки программирования вроде ФОРТРАНа или ПЛ/1, главным, а возможно, и единственным достоинством которых является то, что правильно написанная программа на этих языках использует аппаратуру ЭВМ почти столь же эффективно, что и программа, написанная в машинных кодах. Сегодня ситуация кардинально изменилась. Современные ЭВМ работают все быстрее, а их стоимость падает. Отношение производительности ЭВМ к затратам на ее достижение за 35 лет развития средств вычислительной техники возросло в миллион раз! С другой стороны, стоимость разработки программ неуклонно возрастает, и в основу критериев их качества теперь закладываются иные параметры.

Хорошая программа должна быть быстро создаваемой, легко отлаживаемой. Программа должна быть надежной (т. е. не содержать скрытых ошибок), легко обсуждаемой с заказчиком и быстро ему передаваемой. Программа, наконец, должна быть легко модифицируемой. Ни по одному из этих критериев традиционные языки программирования не наберут много очков. Несмотря на то что некоторые новые машинные языки этого типа частично удовлетворяют перечисленным требованиям (например, ПАСКАЛЬ или МОДУЛА позволяют достичь высокой надежности программирования), они не способны обеспечить главного на сегодняшний день критерия качества — скорости и дешевизны разработки прикладного программного обеспечения.

Число задач растет значительно быстрее, чем пополняются ряды программистской армии. Положение дел будет неизбежно ухудшаться по мере того, как электронная промышленность будет наводнять рынок все более дешевыми и быстродействующими ЭВМ. При современных методах программирования никакое мыслимое количество людей не способно насытить программами растущий поток вычислительной техники. Если ничего не сделать для повышения производительности труда большинства программистов, то уже в начале следующего века ВСЯ рабочая сила промышленно развитых стран будет вынуждена заниматься кодированием программ! По оценкам специалистов, для нормального развития информационной индустрии необходимо увеличить производительность труда программистов в 50—100 раз. Заметим, что за всю историю развития вычислительной техники продуктивность их труда выросла менее чем в 4 раза.

Как же достичь такой производительности программирования? Это можно сделать только с помощью перехода к использованию более мощных языков там, где это возможно. Подобно тому как кустарные производства были вытеснены мануфактурами, автоматизированные средства разработки программ должны заменить низкопроизводительные традиционные машинные языки. Это, впрочем, не означает, что ПАСКАЛЬ, СИ, КОБОЛ и им подобные языки должны быть полностью изъяты из обращения. Напротив. Реализация операционных систем, новых языков, систем управления базами данных и многих других подобных программ потребует полчищ высококвалифицированных программистов, вооруженных традиционными языками. Эти программисты, которых часто называют системными, обречены на использование таких языков в силу специфики своей деятельности.

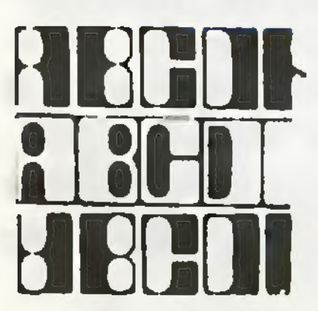
Другое дело — создание прикладных программ. К таким разработкам сейчас предъявляются жесточайшие

требования по времени их реализации и, самое главное, по стоимости. Прикладной программист должен не только уметь кодировать алгоритмы, но и представлять себе ту предметную область, в которой его программа будет использоваться. Применение автоматизированных программных средств и более мощных языков поможет не только повысить производительность труда прикладного программиста, но и не позволит деревьям синтаксических конструкций примитивных языков заслонить леса практических целей.

Большая часть работы по программированию ляжет на плечи непрофессионалов, которых часто называют конечными пользователями. Конечный пользователь — это человек, для которого программирование не является основным видом профессиональной деятельности, а лишь помогает в выполнении его основной работы. Идеальной представляется ситуация, когда конечные пользователи сами программируют свои приложения, заменив, таким образом, прикладных программистов. К сожалению, традиционные машинные языки обладают слишком высоким «порогом освоения», который не в состоянии преодолеть большинство конечных пользователей. Языки программирования, предназначенные для непрофессионалов, должны быть значительно более мощными, чем ПАСКАЛЬ или БЕЙСИК, для того чтобы программистская деятельность не превращалась в рутину.

Всеми упомянутыми свойствами обладает язык программирования АПЛ, который, с одной стороны, очень прост и легко осваивается начинающими, а с другой — совсем не примитивен и способен предоставить простор для творчества самому «матерому» профессионалу. Применение АПЛ обеспечивает десятикратный рост производительности труда по сравнению с такими языками, как ФОРТРАН или ПЛ/1. АПЛ-программы на один-два порядка короче, чем аналогичные, написанные на традиционных языках программирования. Поскольку стоимость программы зависит лишь от длины ее текста и не зависит от используемого языка, то АПЛ-программы во столько же раз дешевле.

Чтобы лучше понять, что же это за язык — АПЛ, давайте сначала вспомним историю.



А. И. МИРОШНИКОВ

ИСТОРИЧЕСКИЙ ОЧЕРК

То, что уже есть, не требует доказательств.

Р. Шекли

История АПЛ мало похожа на историю большинства языков программирования. Она началась еще в 1957 г. трудами Кеннета Айверсона — математика, работавшего в то время в Гарвардском университете (США) и занимавшегося теорией алгоритмов. В своей профессиональной деятельности К. Айверсон, как, впрочем, и все другие алгоритмисты, постоянно сталкивался с проблемой формальной, но в то же время компактной, записи описаний алгоритмов. Словесное описание может быть компактным, но оно не будет математически строгим. С другой стороны, использование для этой цели языков программирования вроде ФОРТРАНа или АЛГОЛа, которые уже были созданы, приводит к тому, что описания даже очень простых алгоритмов становятся слишком громоздкими. Они содержат массу ненужных математику подробностей и совершенно скрывают суть того, что они описывают.

В 1962 г. выходит в свет книга Айверсона, названная им «A Programming Language» («Некоторый язык программирования») — APL.

Книга не имеет никакого отношения к практическому программированию в том смысле, как мы понимаем его сейчас. В ней была предложена элегантная, компактная и строгая формальная алгебраическая нотация — система записи алгоритмов обработки информации. Проблемы, поднятые в этой книге, волновали немногих, и, несмотря на ее огромный успех среди специалистов, никто не предполагал, что 1962 г. войдет в историю информатики как официальная дата рождения самого мощного из известных ныне универсальных языков программирования, а его

«изобретатель» будет награжден международной Тьюринговской премией.

Вскоре Айверсон покидает Гарвард и переходит на работу в один из исследовательских центров фирмы IBM, что расположен в штате Нью-Йорк. В 1964 г. Ф. Фалькоф, К. Айверсон и Е. Сассенгут публикуют формальное описание System/360, используя для этого АПЛ. Эта работа стала самым полным и лаконичным описанием аппаратуры вычислительной машины IBM/360 (в СССР машины этого типа известны как ЕС ЭВМ). Насколько нам известно, больше никогда и никому не удалось повторить подобную работу ни для какого вычислительного устройства сравнимой сложности.

Со свойственным ей размахом фирма IBM начала работы по реализации экспериментальной вычислительной системы, способной интерпретировать язык АПЛ в качестве входного. Сразу три группы программистов включились в эту работу. Первыми добились успеха Л. Брид и Ф. Абрамс из Калифорнийского центра: в 1964 г. они реализовали, пока еще примитивную, АПЛ-систему для ЭВМ IBM 7090. В результате этой работы Филиппом Абрамсом была сформулирована концепция АПЛ-машины. Собственно, это и было рождением АПЛ как языка программирования для ЭВМ.

В 1966 г. А. Фалькоф, Л. Брид, Ф. Абрамс и Р. Лафвелл вместе с Р. Муром в считанные месяцы реализовали терминальную систему с разделением времени для ЭВМ IBM 360/50. Попутно они решили ряд принципиальных проблем построения системного программного обеспечения. Сегодня мы совершенно естественно воспринимаем понятия диалоговой операционной среды, системы разделения времени, интерактивной обработки. Но мало кто знает, что впервые эти понятия практически были реализованы в системе APL/360. Позже А. Роузом был реализован полный интерпретатор с АПЛ, который и вошел в состав системы APL/360. Эта система по праву считается «классической» работой в области программирования. Именно ей АПЛ обязан мгновенной известностью среди программистов. Остается только сожалеть о том, что при «адаптации» программного обеспечения для ЕС ЭВМ она «затерялась» в недрах одного из наших НИИ.

В период с 1966 по 1969 г. IBM упорно отказывалась от коммерческого распространения APL/360, предпочитая использовать эту систему для внутренних целей в качестве «секретного оружия». Отчасти это было вызвано тем, что ею уже были затрачены огромные средства на разработку и рекламу двух новых языков — PL/1 и COBOL. Потребители уже знали эти языки, и IBM предпочла сделать ставку именно на них, не желая тратить средства на рекламу совершенно нового продукта. К тому же руководство фирмы считало, что будущее АПЛ — в научных исследованиях, а ситуация на рынке программного обеспечения требовала появления языков программирования, ориентированных в первую очередь на применение в сфере бизнеса. В эти годы АПЛ распространялся «по знакомству», что и объясняет его несколько запоздалое признание.

В 1968 г. состоялась первая официальная продажа APL/360 научному центру NASA, где он был использован для моделирования системы управления спутниками.

В 1969 г. IBM объявила о поддержке APL/360 на своих ЭВМ систем 360 и 370 и начала его коммерческое распространение. В том же году состоялась первая международная конференция по АПЛ, ставшая затем ежегодной. Конец 60-х — начало 70-х годов стали периодом первой волны повального увлечения АПЛ. CDC, Univac, Burroughs, DEC, Xerox, Honeywell — все выбросили на рынок свои реализации АПЛ-систем. В Европе АПЛ начал использоваться в научных центрах, учебных заведениях и офисах фирм Великобритании, Голландии, Дании, Италии, Испании, Франции. К делу подключились коммерческие организации, сдающие в аренду машинное время: Scientific, Time Sharing Corporates (STSC) и I. P. Sharp Associates (IPSA). В короткое время эти две фирмы смогли создать свои, альтернативные IBM, мощные АПЛ-системы и пакеты прикладных программ для них. IBM, STSC и IPSA к настоящему моменту фактически монополизировали международный рынок АПЛ-продуктов.

В нашей стране мало кому известно, что одним из первых в мире микрокомпьютеров был MCM/70 канадской фирмы Micro Computer Machines. Он вобрал практически все самые передовые достижения электронной техники того времени: микропроцессор Intel 8008, плазменный дисплей на одну строку, портативные накопители на компакт-кассетах. При реальном адресном пространстве всего 16 Кбайт АПЛ-интерпретатор поддерживал виртуальную рабочую область объемом 256 Кбайт, что тоже было впервые. В конце 70-х годов профессор П. И. Медов (университет г. Торонто, Канада) привез в СССР два таких компьютера, которые вдохновили очень многих на изучение АПЛ.

Надо сказать, что идея «чистого» АПЛ-компьютера постоянно владеет умами разработчиков. За MCM/70 последовали MCM/524, MCM/700, MCM*POWER, IBM 5100, IBM 5120, японский Ampere, французские ЭВМ фирмы GIXI и др. Сейчас АПЛ-компьютеры предлагает канадская IPSA и... советский научно-технический кооператив «Диалог» (Обнинск).

Фирма IPSA, которая стала развивать АПЛ на больших машинах и в сетях, в 1973 г. дотянула свою сеть, охватывавшую к тому моменту уже всю Северную Америку, до Европы: первый терминал АПЛ-сети появился в Париже.

После того как первый ажиотаж вокруг АПЛ начал спадать, наглядно проявились просчеты IBM в оценке области его применимости. Дело в том, что, поскольку система APL/360 не предназначалась для деловых

применений, в ней отсутствовали собственная файловая система и средства форматирования информации. Это препятствовало применению АПЛ в самой массовой сфере использования ЭВМ — в бизнесе. В спешном порядке на рынок были выброшены системы APL SV фирмы IBM и APL*PLUS фирмы STSC. Несмотря на это, АПЛ уже не смог составить серьезную конкуренцию КОБОЛу в этой области. Тем не менее в планировании своей финансовой деятельности на АПЛ сделали ставку такие гиганты, как IBM, STSC, Xerox, Shell, Coca-Cola и др.

В конце 70-х годов академик Н. Н. Моисеев, во время командировки в Канаду познакомился с АПЛ. Он был поражен легкостью, с которой его канадские коллеги, используя АПЛ, порождали программы и получали нужные результаты. По его инициативе в Вычислительном центре АН СССР (ВЦАН) С. П. Прохоровым была реализована первая отечественная система APL/БЭСМ-6. В 1978 г. в ЦИПК ГК ИАЭ (г. Обнинск) внедрена в учебный процесс система APL/ВИДЕО для ЕС ЭВМ — адаптированная версия системы APL/360. Работы по реализации советской АПЛ-системы были начаты также в Московском энергетическом институте группой программистов под руководством Е. Т. Семеновой.

В 1979 г. применение АПЛ началось в Ленинграде с помощью мощной ЭВМ Cyber-172, установленной в Ленинградском научно-исследовательском ВЦ АН СССР.

В 1981 г. предложена новая ветвь развития АПЛ — APL2, в которой разрешается использовать структуры произвольного вида, а не только регулярные массивы, как это принято в «каноническом» языке.

В 1982 г. начинается новая волна увлечения АПЛ, связанная с появлением персональных ЭВМ. В настоящее время версии АПЛ существуют для всех типов профессиональных ПЭВМ, вне зависимости от типа микропроцессора и версии операционной системы. Для ПЭВМ типа IBM PC существует сразу несколько реализаций, самой мощной из которых является система APL*PLUS/PC фирмы STSC.

В 1984 г. в ВЦ СО АН СССР (Новосибирск) при разработке системы обработки изображений реализовано подмножество языка АПЛ на комплексе СМ-2/ПС-2000, тем самым был реализован один из первых в мире параллельных интерпретаторов АПЛ. В настоящее время IBM и IPSA активно ведут разработки по созданию АПЛ-систем для многопроцессорных комплексов.

На Международной конференции APL84 предложено расширение набора операторов языка. Если раньше примитивы АПЛ хорошо реализовывали концепцию «одна команда — много данных», то новые операторы позволяют перейти к концепции сетевых ЭВМ: «много команд — много данных».

STSC начала распространение системы STATGRAPHICS, предназначенной для статистической обработки данных и графической интерпретации результатов на ПЭВМ IBM PC. Сейчас этот пакет входит в тройку наиболее удачных в области анализа данных.

В 1985 г. в ВЦ АН СССР создана первая полная отечественная реализация АПЛ-системы для мини-ЭВМ СМ-4. Она стала родоначальницей большого семейства систем для мини-, микро- и персональных ЭВМ с общим названием APL/ВЦАН.

В 1986 г. в г. Обнинске состоялось первое, ставшее затем ежегодным, Всесоюзное совещание-семинар по АПЛ. Формируется, пока неформальная, рабочая группа, целью которой становится популяризация АПЛ в СССР и координация разработок с его использованием.

В 1987 г. придумана и реализована советская АПЛ-машина на базе микро-ЭВМ типа ДВК и «Электроника 60М» (подробно см.: Микропроцессорные средства и системы. — 1989. — № 4).

В 1988 г. технический комитет ISO де-юре утвердил международный стандарт языка АПЛ, который де-факто соблюдался всеми разработчиками. АПЛ — это один из самых стандартизованных языков программирования.

В Обнинске появилась первая советская организация, занимающаяся исключительно производством и распространением АПЛ-продуктов — научно-технический кооператив «Диалог». Его новейшая разработка — мощная система APL/85 для ПЭВМ типа «Электроника 85».

В 1989 г. STSC анонсировала свою принципиально новую разработку APL*PLUS System II для ПЭВМ на базе микропроцессора Intel 80386. В этой версии сняты все ограничения на размеры массивов данных, и объем рабочей области может достигать теоретических 2 Гбайт. Эта работа начинает большую программу STSC по созданию единой переносимой системы Portable APL*PLUS, предназначенной для ЭВМ типа VAX и других компьютеров с операционной системой UNIX, а также для System 370.

Несмотря на то что АПЛ существует уже более 25 лет, он по-прежнему считается новым языком. Уникальное свойство легкости расширения, заложенное еще Айверсоном, позволяет ему всегда оставаться актуальным и популярным.

А. В. КОНДРАШЕВ

ЧТО ТАКОЕ АПЛ?

> Многие вещи нам непонятны не потому, что наши понятия слабы, но потому, что сии вещи не входят в круг наших понятий.
>
> *К. Прутков*

Диалоговый язык программирования сверхвысокого уровня АПЛ позволяет кодировать специфические структуры алгоритмов с той же легкостью, с которой язык алгебры позволяет изображать математические понятия. Компактность языка поразительна — средняя длина АПЛ-программ не превышает 10 строк! Использование АПЛ резко повышает производительность труда программиста как по сравнению с компилируемыми языками типа ПЛ/1 или ПАСКАЛЬ, так и по сравнению с другими интерпретируемыми вроде БЕЙСИКа. АПЛ позволяет программисту сконцентрировать все свои силы на том, *что* он хочет сделать с помощью ЭВМ в отличие от традиционных языков, которые постоянно ставят его перед проблемой, *как* это запрограммировать.

Язык АПЛ и АПЛ-система

Язык АПЛ — это некоторая формальная нотация, предназначенная для записи алгоритмов обработки информации. На языке АПЛ можно писать карандашом на бумаге или мелом на доске, не имея даже вычислительной машины. Многие математики используют его именно таким образом.

АПЛ-система — это специальная программа, которая позволяет человеку общаться с ЭВМ на языке АПЛ. АПЛ-система — это, как сейчас принято говорить, диалоговая интегрированная среда, которая включает в себя все необходимые пользователю средства управления ЭВМ, собственную концепцию организации диалога, технологию разработки, отладки, хранения и эксплуатации программ.

Представьте себе, что перед вами находится чистый лист бумаги, карандаш и микрокалькулятор, а в ящиках вашего стола хранятся уже сделанные вами ранее записи, расчеты или чертежи. На книжной полке стоят учебники и справочники. Обычная обстановка, в которую ежедневно попадают инженеры, научные работники, студенты или школьники. Точно так же устроена и АПЛ-система, с той лишь разницей, что роль листа бумаги исполняет память ЭВМ, и размер этого листа, который в АПЛ называется рабочей областью, может достигать нескольких мегабайт. Роль карандаша исполняет клавиатура дисплея. Экран дисплея служит средством связи между АПЛ-системой и человеком. На нем можно увидеть содержимое объектов, хранящихся в рабочей области, или получить ответ АПЛ-системы на заданный ей вопрос. Возможности нашего «микрокалькулятора» совпадают с возможностями ЭВМ. Заметки, сделанные ранее, книги и другие информационные материалы, необходимые для работы, — это АПЛ-библиотеки и файлы данных. Пользователь АПЛ-системы всегда может скопировать в рабочую область любую программу или массив данных, подобно тому как вы заглядываете в справочник и переписываете себе на лист бумаги нужную формулу, набор чисел или некоторый текст.

В рабочей области АПЛ-системы может находиться столько различных программ и массивов данных, насколько это позволяет ее размер. Программы и данные совершенно независимы, точно так же, как одна и та же переменная может входить в различные математические формулы.

Лаконичность языка делает процесс общения с АПЛ-системой очень продуктивным, поскольку время, затрачиваемое на ввод информации, практически равно нулю в отличие, скажем,

от БЕЙСИКа, который заставляет программиста расходовать немало сил на «набивку» даже очень простых программ. С помощью ввода всего лишь одной строки АПЛ-система позволяет решать задачи, которые потребовали бы программ в десятки или даже сотни строк, написанных на традиционных языках.

Элементарные возможности АПЛ-системы

АПЛ очень прост в освоении. Трехдневный курс АПЛ равносилен шестимесячному курсу КОБОЛа! Вы, уважаемый читатель, уже владеете некоторыми его азами. Без сомнения, вам понятно, что означает запись 2×2, а ведь это есть законченная и полностью готовая к исполнению программа на языке АПЛ! Если ввести это выражение с клавиатуры дисплея, работая в АПЛ-системе, и нажать клавишу ВК (возврат каретки), то на экране, как и следовало бы ожидать, появится:

```
    2×2
4
```

Когда курсор на экране дисплея находится в 7-й позиции текущей строки, то это означает, что АПЛ-система готова к приему очередной команды пользователя. Фигурально выражаясь, каждый новый вопрос пользователя начинается с «красной строки». Заметим, что пока не нажата клавиша ВК, вы можете произвольным образом редактировать строчку на экране. В дальнейшем мы не будем специально обращать ваше внимание на необходимость нажатия ВК при вводе. Ответы АПЛ-системы всегда начинаются с первой позиции следующей строки. Это позволяет наглядно видеть на экране весь процесс диалога. Любой человек может сразу же, без всякого специального обучения использовать АПЛ-систему в качестве калькулятора. При этом, естественно, разрешается вводить арифметические выражения любой сложности. Например:

```
    (3×4)+5
17
```

Внимательный читатель может сказать: «Зачем же здесь нужны скобки? Ведь операция умножения сама собой должна использоваться первой!» Однако в АПЛ это не так. В этом языке все операции равноправны, а выражения исполняются справа налево. На это есть весьма веские причины. Во-первых, программисту не надо помнить приоритеты функций, входящих в выражение. Для АПЛ это очень важно, так как количество одних только встроенных функций достигает 1000. Представьте себе, как трудно было бы их использовать в длинном выражении! Ведь тогда пришлось бы постоянно думать о том, надо заключать часть выражения в скобки или нет? Во-вторых, вычисление выражений справа налево, как и отсутствие приоритетов у функций, совпадает с принципом, принятым в высшей математике. Например, математическое выражение $f(g(x))$ на АПЛ будет выглядеть точно так же, при этом скобки даже можно опустить: F G X. Таким образом, если мы повторим ввод нашего выражения, опустив скобки, то получим 27.

Самые элементарные встроенные функции АПЛ называются примитивами. Их в АПЛ около 80. Каждый примитив обозначается определенным значком. Поскольку примитивов много, то в язык введено некоторое количество специальных символов, которых нет в других языках программирования. Среди этих символов, например, есть и уже использовавшийся нами значок операции умножения «×», который отсутствует на клавиатуре обычного дисплея. Умножение и сложение — это примитивы. Наряду с ними в АПЛ есть вычитание, деление, возведение в произвольную степень, логарифмы, тригонометрические функции, максимум, минимум, различные округления и многое другое.

Примитивы могут быть одноместными (с одним аргументом) или двухместными (с двумя аргументами). Единственный аргумент одноместного примитива располагается справа от значка, его обозначающего, а аргу-

менты двухместного — по обе стороны от него. Табл. 1, в которую сведены некоторые примитивы АПЛ, отвечающие за раздел элементарной математики (всего их более 30), и примеры, приведенные ниже, показывают некоторые вычислительные возможности АПЛ.

Таблица 1

Математика	АПЛ	Пояснения
1/x	÷X	Обратная величина
−x	−X	Изменение знака
|x|	|X	Абсолютная величина
sign x	×X	Знак
e^x	*X	Экспонента
ln x	⍟X	Натуральный логарифм
Lxl	LX	Округление вниз
Гxl	ГX	Округление вверх
x! или Г(x)	!X	Гамма-функция
πx	oX	Умножение на «пи»
$\bar{x}$	~X	Логическое отрицание
x−y	X−Y	Вычитание
x/y	X÷Y	Деление
min (x, y)	XLY	Минимум
max (x, y)	XГY	Максимум
x mod y	YIX	Остаток
x^y	X*Y	Возведение в степень
$\log_y x$	Y⍟X	Логарифм
C_y^x	X!Y	Сочетания
x≥y	X≥Y	Больше или равно
xvy	XVY	Логическое «ИЛИ»
xΛy	XΛY	Логическое «И»

```
  ÷4
0.25
  1.5÷2.4
0625
  10L20
10
  Г4.5
5
  L4.5
4
  318
2
  5≥7
0
  (5≥7) V 5<7
```

То, что для обозначения некоторых примитивов в АПЛ используются специфические значки, отсутствующие на клавиатуре стандартного дисплея, долго служило серьезным сдерживающим фактором широкого распространения АПЛ-систем. Однако с появлением терминалов, набор символов которых можно программировать, эта преграда была устранена раз и навсегда. Для отечественных ЭВМ также найдены удовлетворительные способы решения этой проблемы.

До сих пор мы говорили о том, как использовать АПЛ-систему в режиме «счета на промокашке», когда результаты работы лишь видны на экране и нигде не запоминаются. Наш «лист» — рабочая область — все еще пуст, как и в начале работы. Для запоминания произвольной информации в языке АПЛ используется примитив «спецификация», который обозначается символом «←».

```
  A←2
```

Это АПЛ-выражение эквивалентно фразе: «Пусть А равно 2». Где и в каком виде будет храниться объект с именем А, решает сама АПЛ-система. Пользователь полностью освобождается от необходимости заранее резервировать место для его хранения и объявлять тип объекта. Все это делает АПЛ-система автоматически. Для нас важно лишь то, что теперь достаточно ввести с клавиатуры имя, и система сразу же покажет на экране значение, которое за ним скрывается:

```
  A
2
```

Вы, очевидно, обратили внимание на то, что когда мы попросили АПЛ-систему запомнить число 2 под именем А, на экране ничего больше не появилось. Так бывает всегда, когда последняя исполняемая функция строки — спецификация.

Значение, обозначенное некоторым именем, можно использовать в вычислениях. Например:

```
  10—A
8
```

Спецификация является таким же обычным примитивом, как и все остальные. Она может находиться в произвольном месте выражения:

```
  5+B←A×8
21
  B
16
```

Разрешается также изменять значение переменной во время исполнения одной строки, как в этом случае:

```
  A
2
  A+A←A+3
10
  A
5
```

Обратите внимание на то, что при выполнении последней операции сложения было использовано уже новое значение переменной А. Имена в АПЛ могут состоять из любого количества букв и цифр. Важно лишь, чтобы имя всегда начиналось с буквы.

Массивы данных

АПЛ лучше других языков приспособлен для обработки массивов данных. Он может работать с линейными массивами произвольных размерностей. Одномерные массивы, называемые векторами, можно использовать в качестве констант в выражениях языка. Например:

```
  ВЕКТОР←1  2  3  4  5
  ВЕКТОР
1  2  3  4  5
```

Вектор определяется перечислением всех его элементов. В числовых векторах один элемент от другого отделяется пробелом. В АПЛ можно определить и символьный вектор, т. е. вектор, элементами которого будут не числа, а литеры. Символьный вектор определяется вводом нужной строки литер, слева и справа ограниченной кавычками. Естественно, что в символьном векторе пробелы равноправные его члены. Например:

```
  ТЕКСТ←'ЭТО СТРОКА'
  ТЕКСТ
ЭТО СТРОКА
```

Все примитивы, рассмотренные нами выше, называются скалярными. Это означает, что они могут быть использованы в операциях с массивами точно так же, как они используются в операциях со скалярами (одиночными числами). Скалярные примитивы действуют на массивы поэлементно. (Математики говорят, что скалярные примитивы обладают свойством индексной дистрибутивности.) Давайте попробуем сложить два вектора:

```
  1  2  3 + 4  5  6
5  7  9
```

Здесь операция сложения была выполнена над каждой парой, составленной из соответствующих элементов двух векторов. Это и есть индексная дистрибутивность. Операция как бы «просачивается» через массивы, совершая заданное действие над их элементами, имеющими одинаковые номера (индексы). Очевидно, для того чтобы это было возможно, векторы, участвующие в операции, должны иметь одинаковую длину. Однако в АПЛ разрешается в качестве любого из аргументов скалярного примитива использовать скаляр вместо вектора. В этом случае АПЛ-система «вытягивает» скаляр в вектор подходящей длины путем его многократного повторения. Например, если мы хотим умножить все элементы массива ВЕКТОР на 5, мы можем поступить так:

```
  5  5  5  5  5×ВЕКТОР
5  10  15  20  25
```

Но гораздо удобнее и проще написать просто:

```
  5×ВЕКТОР
5  10  15  20  25
```

Точно так же можно работать и с символьными векторами. Например, выясним, на каких местах в векторе ТЕКСТ находятся буквы «Т» или «О»?

```
    ТЕКСТ='Т'
0  1  0  0  0  1  0  0  0  0
    ТЕКСТ='О'
0  0  1  0  0  0  0  1  0  0
  (ТЕКСТ='О') ∨ ТЕКСТ='Т'
0 1 1 0 0 1 0 1 0 0
```

Вы, вероятно, поняли, что логические примитивы выдают в качестве результата 1, когда реализуется ответ «ДА», и 0 — в противоположном случае. Мы сначала сравнили массив ТЕКСТ со скаляром «Т», а затем со скаляром «О». Дизъюнкция полученных таким образом векторов дает окончательный результат, в котором 1 отмечает те позиции вектора ТЕКСТ, в которых находятся литеры «Т» или «О».

А что будет, если мы ошибемся и используем в операции векторы разной длины?

```
  4  5  6—2  3
4  5  6 — 2  3
        ∧
ОШИБКА: РАЗМЕР
```

Всегда, когда АПЛ-система обнаруживает ошибку, она распечатывает на экране текст ошибочной строки и указывает место и характер ошибки. В данном случае она сообщила нам, что вычитание не может быть выполнено, так как размеры векторов не согласованы: для одного элемента левого вектора нет ему соответствующей пары из правого. Выдача сообщения об ошибке не приводит к каким-либо катастрофическим последствиям. Пользователь всегда имеет возможность продолжить работу после исправления ошибки.

Скалярными могут быть не только двухместные примитивы, но и одноместные (см. табл. 1). Например, изменим знаки всех элементов массива ВЕКТОР на противоположные:

```
  -ВЕКТОР
¯1 ¯2 ¯3 ¯4 ¯5
```

Для того чтобы можно было отличать значок операции «изменение знака» от отрицательного знака числа, в АПЛ для обозначения отрицательных чисел используется литера «надчерк». Следующий пример наглядно показывает, для чего это необходимо:

```
  –¯2  3  ¯5  7
2  ¯3  5  ¯7
```

В качестве самостоятельного упражнения определите, что за числа получаются в результате ввода следующей строки:

```
  0.25 0.5 1 2
0.7854 1.5708 3.1416 6.2832
```

Таким образом, мы познакомились с первыми двумя свойствами языка АПЛ, которые делают его таким мощным: большое число встроенных функций и возможность работать с массивами данных точно так же, как и с отдельными числами. В других языках программирования нам пришлось бы заниматься кодированием циклов, чтобы получить подобные результаты.

Кроме скалярных примитивов, которые в основном покрывают аппарат элементарной математики, в АПЛ имеются так называемые смешанные примитивы. Для них уже не выполняется свойство индексной дистрибутивности, и правила их использования с массивами индивидуальны. Сейчас нам понадобится лишь один из них, который называется «перестройка» и обозначается греческой буквой «⍴»

Перестройка — это двухместный примитив, который служит для формирования массивов произвольной размерности и структуры. Левый аргумент должен быть вектором целых чисел. Каждый элемент этого вектора показывает длину соответствующей оси массива-результата. Правый аргумент — это «склад» значений, из которых будет строиться результат. Этот аргумент может быть совершенно произвольным массивом. Если оказывается, что для построения массива-результата требуется больше элементов, чем их имеет правый аргумент, то по мере их исчерпания перестройка циклически повторяет правый аргумент столько раз, сколько это необходимо. Давайте построим матрицу размерами 3×4, состоящую из первых 12 чисел натурального ряда:

```
  M←3 4⍴1 2 3 4 5 6 7 8 9 10 11 12
  M
1  2  3  4
5  5  7  8
9  10 11 12
```

А в этом случае построим символьную матрицу, состоящую из символов «*»:

```
  S←3 7⍴'*'
  S
.......
.......
.......
```

Работа с матрицами и массивами бóльших размерностей осуществляется точно так же, как и с векторами. Необходимо лишь заботиться о том, чтобы размеры массивов были согласованы, т. е, длины их соответствующих измерений совпадали. Выясним, например, на каких местах в матрице М находятся элементы, большие 6:

```
  M>6
0 0 0 0
0 0 1 1
1 1 1 1
```

Используя табл. 1, поясните смысл следующих действий:

```
  3IM
1 2 0 1
2 0 1 2
0 1 2 0
  2*M
  2        4        8       16
 32       64      128      256
512     1024     2048     4096
  6LM
1 2 3 4
5 6 6 6
6 6 6 6
```

Язык АПЛ имеет очень мощную функцию индексирования, которая по своим возможностям намного превосходит аналогичные операции из других языков программирования. Если в большинстве языков программирования с помощью индексирования можно извлечь из массива данных только один элемент с указанными индексами, то АПЛ предоставляет практически неограниченные возможности по извлечению любых регулярных подмассивов из данного массива. Например:

```
  M[2;3]
7
  M[2;3 2]
7 6
  M[;3]
3 7 11
  M[3 1;3 4]
11 12
3  4
```

Как видно из этих примеров, в АПЛ разрешено использовать в качестве индексов не только скаляры (первый пример), но и векторы (второй и четвертый примеры). Если какой-либо индекс пропущен, как в третьем примере, то выбираются все подмассивы вдоль этого измерения индексируемого массива. В третьем случае из всех строчек матрицы M были выбраны элементы, находящиеся в столбце 3. В качестве индекса могут быть использованы и матрицы и массивы любой другой размерности. При этом результат индексирования будет иметь такую же структуру, что и индекс. Это свойство индексирования можно, например, использовать для наглядного представления информации. Предположим, в некотором тексте необходимо отметить все буквы «А». Это можно сделать так:

```
  ИМЕНА←3 4 ⍴'МАША  ТАНЯ  СЕВА'
  ИМЕНА
МАША
ТАНЯ
СЕВА
  ИМЕНА='А'
0 1 0 1
0 1 0 0
0 0 0 1
  1+ИМЕНА='А'
1 2 1 2
1 2 1 1
1 1 1 2
  '.*'[1+ИМЕНА='А']
.*.*
.*..
...*
```

Для решения поставленной задачи достаточно было ввести лишь последнюю строчку, но мы специально выполнили все действия по шагам для большей наглядности. АПЛ-система выполняет все действия точно в такой же последовательности.

Операторы

Огромную роль в математике играют операторы (не путать с операторами языков программирования!) Если функции, преобразуют одни данные в другие, то операторы, действуя на функции, создают новые функции — производные функции. Есть операторы и в АПЛ. Мы познакомимся лишь с некоторыми из них.

Первый оператор называется редукция. Смысл его работы можно легко понять, если вспомнить, как «устроены» такие, например, математические выражения:

$$\sum_i x_i \quad \prod_i x_i \quad \min_i x_i \quad \max_i x_i$$

Несмотря на то что они обозначаются по-разному, принцип их действия одинаков: берется какая-либо двухместная функция и расставляется между членами массива, а затем полученное выражение вычисляется. Таким образом, вместо вектора получается одно число — вектор редуцируется (сжимается) в скаляр с помощью двухместной функции. В АПЛ такой оператор

обозначается символом "/". Он может быть применен к любому двухместному скалярному примитиву. Значок примитива, к которому применяется оператор редукции, указывается слева от "/". Испытаем работу этого оператора на массиве ВЕКТОР.

```
  ВЕКТОР
1 2 3 4 5
  +/ВЕКТОР
15
  ×/ВЕКТОР
120
  L/ВЕКТОР
1
  Г/ВЕКТОР
5
```

Эти строчки, как легко догадаться, вычисляют сумму и произведение элементов массива ВЕКТОР, а также находят значения его минимального и максимального элементов. Если бы мы использовали БЕЙСИК или ПАСКАЛЬ, то без программирования циклов мы бы не обошлись, да и вместо одной операции пришлось бы писать целую программу! В то же время выражения АПЛ практически совпадают с математической записью. Обратите внимание на то, что нигде не указана длина массива, ведь АПЛ-системе она всегда известна! Таким образом, все эти выражения будут одинаково хорошо работать на векторах произвольной длины. Во-первых, это значительно разгружает тексты АПЛ-программ, поскольку нет нужды в использовании «паразитических» переменных, обозначающих параметры циклов, и различных «ритуальных» слов, вроде DO, NEXT, FOR и им подобных. В АПЛ вообще не используются слова из естественных языков, и в этом смысле он интернационален, как и математика. Во-вторых, такие свойства АПЛ дают возможность писать программы, не зависящие от структуры данных, что очень важно во многих приложениях.

Теперь, вооруженные таким мощным средством, мы можем решать более сложные задачи. (Если вы уже знакомы с каким-либо языком программирования, то попробуйте сделать то же самое на нем.) Например, сколько элементов в массиве ВЕКТОР имеют значение, большее 3?

```
  +/ВЕКТОР>3
2
```

Попробуйте самостоятельно определить, на какие вопросы отвечают следующие выражения АПЛ, в которых значками «V» и «Λ» обозначены примитивы «логическое ИЛИ» и «логическое И»:

```
  V/ВЕКТОР>3
1
  Λ/ВЕКТОР>3
0
  Λ/ВЕКТОР>0
1
```

Редукция может быть применена не только к векторам, но и к массивам других размерностей. Выполним плюс-редукцию нашей матрицы М:

```
  +/M
10 26 42
```

Как объяснить полученный результат? Вообще говоря, редукцию матрицы можно выполнить по одному из двух измерений — вдоль строк (сумма по столбцам) и вдоль столбцов (сумма по строкам). По умолчанию, как в этом примере, редукция действует по последнему измерению. Таким образом, мы нашли суммы строк матрицы М. АПЛ позволяет явно указать измерение, вдоль которого должна выполняться редукция массива. Найдем суммы столбцов матрицы М:

```
  +/[1]M
15 18 21 24
  +/+/M
78
```

Последняя строка вычисляет сумму всех элементов М. Не правда ли, АПЛ-выражение +/+/М очень похоже на то, что в таких случаях пишут в математике $-\sum_i\sum_j M_{ij}$?

Задача о среднем значении вектора чисел, очень часто встречающаяся в приложениях, на АПЛ может быть запрограммирована так:

```
  (+/ВЕКТОР)÷⍴ВЕКТОР
3
```

Здесь символ «÷» обозначает операцию деления, а уже знакомый нам значок «⍴» — это символ одноместного примитива «структура». Если двухместный примитив «перестройка» служит для формирования массивов, то

«структура» выдает в качестве результата своей работы вектор целых чисел, которые показывают длины измерений массива—аргумента. Понятно, что сколько элементов содержит ⍴X, такова размерность X. В нашем случае ВЕКТОР — это одномерный массив, значит, ВЕКТОР — это одно число, равное количеству элементов в массиве ВЕКТОР. Нам опять удалось написать программу, которая не содержит циклов и является независимой от размера исходного массива! (Вообще, циклы в АПЛ встречаются не слишком часто.) Итак, мы просто выразили на АПЛ словесное описание правила вычисления среднего значения: «Подсчитать сумму элементов массива и поделить на их количество».

Конечно, было бы не слишком удобным каждый раз, когда нам необходимо вычислить среднее значение, вводить с клавиатуры такую длинную строку. АПЛ-система позволяет значительно упростить этот процесс и сделать его более наглядным. Здесь мы временно прервем разговор об операторах, чтобы остановиться на одном, самом ключевом моменте языка АПЛ.

Определяемые пользователем

Любое количество строк, написанных на языке АПЛ, можно объединить в специальный объект — функцию, определяемую пользователем. Для краткости их называют просто определяемыми функциями. По своему использованию определяемые функции ничем не отличаются от примитивов или производных функций. С той лишь разницей, что примитив обозначается специальным символом, а определяемая функция имеет имя, задаваемое программистом. АПЛ-система предоставляет мощные и удобные средства по созданию и отладке определяемых функций. Однако подробный разговор об этом выходит за рамки нашего сегодняшнего знакомства с АПЛ.

Если примитивы — это кирпичики для сооружения здания строки, которую мы вводим с клавиатуры для решения той или иной проблемы, то определяемые функции — это уже целые строительные блоки, которые могут быть сделаны сколь угодно крупными. В АПЛ обычно применяется «крупнопанельное строительство». Программист, работающий на АПЛ, может однажды создать определяемую функцию, а затем вообще забыть о том, как она устроена — лишь бы были известны правила, по которым она преобразует аргументы в результат. Все пользователи АПЛ применяют в своей работе не только свои собственные определяемые функции, но и функции, написанные другими программистами. Очень часто работа непрофессионала сводится к манипулированию уже готовыми определяемыми функциями из некоторого пакета. (Вспомните, как вы пользуетесь готовыми формулами из справочников при выполнении своей работы.)

По числу аргументов определяемые функции делятся на нуль-местные (без аргументов), одноместные и двухместные. Имена определяемых функций могут без ограничений использоваться в АПЛ-выражениях, перемежаясь с примитивами или производными функциями. Внутри определяемой функции могут быть обращения к другим определяемым функциям или даже к самой себе. В АПЛ нет никаких ограничений на глубину вложения функций.

Вот как будет выглядеть определяемая функция, вычисляющая среднее значение числового массива:

```
  R←СРЕДНЕЕ A
[1] R←(+/A)÷⍴A
```

Что все это означает? Каждая определяемая функция имеет служебную строку — заголовок, где задаются правила ее использования среди других функций. В нашем случае функция названа СРЕДНЕЕ. Она имеет один аргумент, обозначенный в ее теле буквой A, и выдает явный результат, обозначенный буквой R. Какие буквы использовать для обозначения аргумента и результата — безразлично. «A» — это так называемый формальный параметр, который получит конкретное значение лишь при вызове

функции. Термин «явный результат» означает, что результат работы функции может быть затем непосредственно использован в качестве аргумента другой функции. Остальные строки функции определяют процесс преобразования аргументов в ее результат — тело функции. Тело функции СРЕДНЕЕ состоит из одной строки. При распечатке текста функции АПЛ-система автоматически нумерует строки ее тела для удобства восприятия. Когда программист определяет функцию, он, конечно же, не нумерует ее строчки, как это принято в БЕЙСИКе.

Исполнение определяемой функции начинается, когда ее имя встречается в строке, исполняемой АПЛ-системой, и все аргументы функции уже вычислены. Это очень похоже на то, как «оживает» аналитическая формула, когда вы подставляете в нее вместо буквенных обозначений конкретные числа. Исполнение функции происходит так же, как если бы мы вводили строчки, образующие ее тело, непосредственно с клавиатуры.

Давайте испытаем работу нашей функции.

```
  СРЕДНЕЕ 1 3 5 7 9
5
  СРЕДНЕЕ ВЕКТОР
3
```

Работа стала значительно удобнее. Теперь вообще можно забыть о том, как именно вычисляется среднее значение. С этого момента об этом будет заботиться АПЛ-система всякий раз, когда ей встретится термин СРЕДНЕЕ. Таким образом, мы расширили стандартный АПЛ еще на одну функцию. А вот более сложный пример:

```
  СРЕДНЕЕ (ВЕКТОР—СРЕДНЕЕ ВЕКТОР)*2
2
```

Эту строчку можно легко «прочитать»: вычислить среднее значение квадратов отклонений элементов массива ВЕКТОР от среднего. Вот вам еще один довод в пользу вычисления выражений справа налево!

Определяемые функции — это и есть АПЛ-программы. В рабочей области может находиться столько таких программ, на сколько позволяет ее объем, в отличие от БЕЙСИК-системы, которая способна работать лишь с одной программой.

Любая строка определяемой функции (кроме заголовка) может быть помечена меткой — некоторым именем, предшествующим тексту этой строки, и отделенным от него двоеточием. Значение метки равно номеру строки, которую она помечает. Значение метки можно использовать в вычислениях. Однако АПЛ-система не допускает возможности применения примитива спецификации к метке. Имена меток локализованы внутри определяемых функций, т. е. значение, связанное с именем метки, существует только тогда, когда исполняется данная определяемая функция. Кроме меток, внутри определяемых функций локализованы и все имена, которые встречаются в заголовке. Например, в нашей рабочей области мы уже определили имя «А», которое совпадает с именем аргумента функции СРЕДНЕЕ (см. ее текст). При вызове функции старое значение А будет временно потеряно и внутри функции именем А будет обозначен ее массив-аргумент. Когда исполнение завершится, АПЛ-система восстановит то значение А, которое существовало до вызова функции.

Если не указано противное, строчки функции исполняются в порядке возрастания их номеров. Порядок исполнения строк можно изменить с помощью примитива «переход», обозначаемого значком →. Это одноместный примитив. Его единственный аргумент может быть произвольным числовым массивом, первый элемент которого — целое число, которое трактуется АПЛ-системой как номер следующей исполняемой строки. Если строки с таким номером нет в теле функции, то исполнение функции завершается. Для формирования аргумента примитива перехода могут быть использованы любые средства языка. Поэтому возможности этого одного примитива покрывают возможности любых мыслимых конструкций организации ветвления, имеющихся во всех

остальных языках программирования. Ниже приведены некоторые примеры использования примитива перехода.

Безусловные переходы:

→M (на строку с метной M)
→3 (на строку с номером 3)

Условные переходы:

→(A<0)/M (на строку с меткой M, если A<0)
→(M1, M2, M3) [2+×A]
(на одну из строк, в зависимости от знака A)
→(N≥I←I+1)/M (переход типа FOR—I—NEXT)
→IΦL1, L2, L3,... (переход по счетчику I)
→(V=0)/0 (выход из функции, если V=0

Операторы (продолжение)

Однако вернемся к нашим операторам! Второй оператор, который называется «внутреннее произведение», также имеет «родственника» в математике. Это хорошо известное скалярное произведение. Каждый из вас, кто изучал высшую математику, даже в минимальном объеме, наверняка помнит «магическую формулу» умножения матриц: «для того чтобы получить элемент с индексами *i* и *j* матрицы результата, надо взять из левой матрицы строчку с номером *i* и почленно умножить ее на столбец с номером *j* из правой матрицы, а полученные числа сложить». Уф! Итак, имеются две операции, первая из которых выполняется почленно над строчкой и столбцом, а вторая — редуцирует полученный таким образом вектор в одно число. Внутреннее произведение АПЛ — это оператор, позволяющий объединить любые два двухместных скалярных примитива в единую функцию, которая и будет работать по такому правилу.

Обозначается этот оператор значком «.» (точка). Справа от него располагается значок примитива, выполняемого первым, а слева — вторым. Например, функция, реализующая обычное матричное умножение, будет выглядеть как "+.×". Найдем скалярный квадрат массива ВЕКТОР.

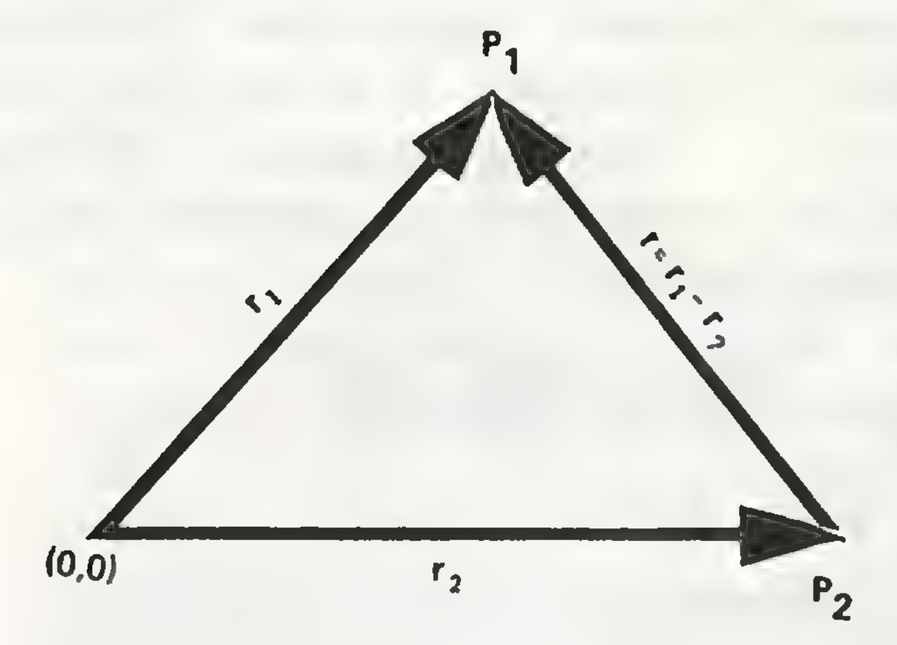
```
  ВЕКТОР+.×ВЕКТОР
55
  (ВЕКТОР+.×ВЕКТОР)*0.5
7.4162
```

Последнее выражение дает нам длину ВЕКТОРа в 5-мерном пространстве. Внутреннее произведение способно работать с массивами произвольных размерностей. Важно лишь, чтобы длина последнего измерения левого массива совпадала с длиной первого измерения правого (вспомните правило умножения матриц). Другими словами, два массива «сворачиваются» в один по их внутренним измерениям. Отсюда и название этого оператора. Всего в АПЛ можно породить около 400 скалярных произведений.

К внутреннему произведению сводится огромное количество задач, встречающихся в приложениях: поиск данных, кодирование информации, анализ сетей и графов и многое другое. Рассмотрим лишь некоторые из них.

Как известно, расстоянием между двумя точками называется квадратный корень из скалярного квадрата вектора, являющегося разностью радиусов векторов этих точек:

$$d=\sqrt{(\vec{r}_1-\vec{r}_2)\cdot(\vec{r}_1-\vec{r}_2)}.$$



Используя эту формулу, мгновенно напишем двухместную определяемую функцию РАССТОЯНИЕ, которая в качестве своего явного результата выдает расстояние между точками, координаты которых задаются левым и правым ее аргументами.

```
      R←R1 РАССТОЯНИЕ R2
[1] R←(R2—R1)+.×R2—R1)*0.5
      1 5 РАССТОЯНИЕ 7 1
7.2111
      1 3 6 РАССТОЯНИЕ 7 3 9
6.7028
```

Построенная нами функция может работать без изменений в пространствах произвольных размерностей.

Пусть есть некоторая матрица — безразлично символьная или число-

вая. Каждая ее строка представляет собой набор значений некоторых признаков. Пусть имеется какой-то вектор. Необходимо выяснить, есть ли среди строк матрицы такая, которая совпадала бы с предъявленным образцом. Решить эту задачу также можно с помощью внутреннего произведения, но уже сконструированного из примитивов «логическое И» и «равно». Действительно, для того чтобы узнать, совпадает ли некоторая строчка матрицы с образцом, необходимо сначала почленно сравнить все элементы строки и образца, а затем, сделав «И»-редукцию, выяснить, все ли элементы совпали.

```
      M
1   2   3   4
5   6   7   8
9  10  11  12
   M∧.=5 6 7 8
0 1 0
```

В результате такого «умножения» матрицы на вектор получаем вектор, имеющий 1 на тех местах, которые соответствуют строчкам, совпадающим с образцом, и 0 на остальных местах. Определяемая функция ПОИСК позволяет решать эту задачу более удобно.

```
      MAT  ПОИСК  BEK
[1] (2 4 ⍴'НЕТ ЕСТЬ')[1+MAT ∧.=BEK;]

  M ПОИСК 5 6 7 8
ЕСТЬ
  M ПОИСК 4 3 2 1
НЕТ
  ИМЕНА ПОИСК 'СЕВА'
ЕСТЬ
```

Функция ПОИСК — это функция без явного результата, поскольку в ее заголовке отсутствует конструкция вида «R←». Ответ, который выдается на экран, — это результат работы ее строки (вспомните, что когда строчка не заканчивается присваиванием, то АПЛ-система выдает результат исполнения строки на экран), а не самой функции, как это было во всех предыдущих примерах. Функция, подобная этой, может быть только самой левой в цепочке суперпозиций функций, образующих АПЛ-выражение.

От функции ПОИСК уже недалеко до простой базы данных. Пусть имеется числовой вектор В, элементы которого это возраст лиц, имена которых перечислены в строках матрицы ИМЕНА.

```
    B←25 21 27
    B
25 21 27
    ИМЕНА
МАША
ТАНЯ
СЕВА
```

Напишем функцию ВОЗРАСТ, которая выдает возраст лица, имя которого указано в качестве ее аргумента.

```
    ГОД←ВОЗРАСТ ИМЯ
[1] ГОД←(ИМЕНА∧.=ИМЯ)/B
    ВОЗРАСТ 'МАША'
25
    ВОЗРАСТ 'СЕВА'
27
```

Мы использовали новый двухместный примитив АПЛ, который нам раньше не попадался. Он обозначен значком «/» и называется «сжатие». Несмотря на то что «сжатие» обозначается так же, как и оператор редукции, конфликтов не возникает. Когда «/» обозначает оператор редукции, тогда слева от него обязательно находится значок скалярного примитива, а когда примитив «сжатие», тогда некоторое выражение. «Сжатие» сжимает свой правый аргумент по шаблону, заданному его левым аргументом. Вернитесь немного назад и вспомните, что получилось в результате операции M∧.=5 6 7 8. Вот вектор такого вида и используется в качестве шаблона: результат сжатия состоит только из тех элементов правого аргумента, которым соответствует 1 в векторе-шаблоне. Модифицируем функцию ВОЗРАСТ, с тем чтобы ею было удобнее пользоваться.

```
    ВОЗРАСТ ИМЯ; ГОД
[1]→M×⍳⍴ГОД←(ИМЕНА∧.=ИМЯ)/B
[2] 'В картотеке нет человека'; ИМЯ
[3]→0
[4]M:ИМЯ; 'имеет возраст'; ГОД
    ВОЗРАСТ 'ТАНЯ'
ТАНЯ имеет возраст 21

    ВОЗРАСТ 'ПЕТЯ'
В картотеке нет человека ПЕТЯ
```

Обсудим некоторые моменты. Во-первых, как вы должны были заметить, в теле функции ПОИСК мы использовали массивы ИМЕНА и В, которые не являются аргументами функции, а существуют вне ее. Определяемые функции АПЛ «непрозрачны» лишь в одну сторону: никакой локальный объект функции недоступен вне ее, но любой объект, существующий вне функции (в том числе и в теле функции, которая вызвала данную), доступен для использования внутри нее. Это позволяет сделать функции независимыми от массивов информации. Во-вторых, в заголовке функции нами объявлена локальной переменная ГОД, которая не является ни аргументом, ни результатом, а служит для хранения промежуточного результата. Мы сделали это для того, чтобы не засорять рабочую область. Когда исполнение функции завершится, эта переменная перестанет существовать. В строке функции осуществляется переход на строку 4, когда предъявленное имя найдено в матрице-картотеке. Выражение «→0» всегда приводит к выходу из функции.

Завершая разговор об операторах языка АПЛ, остановимся еще на одном простом, но очень полезном операторе, он называется «внешнее произведение». Его аналог в математике — декартово произведение множеств. С внешним произведением все мы знакомимся еще в начальной школе, когда учим таблицу умножения. Что такое таблица умножения? Это два вектора чисел от 1 до 10, из элементов которых составляются все возможные пары, и в каждой паре осуществляется операция умножения. Затем все результаты собираются в таблицу (матрицу). Оператор внешнего произведения АПЛ работает точно так же, с той лишь разницей, что он может быть применен к любому бинарному скалярному примитиву, и эта новая производная функция может иметь своими аргументами массивы произвольной размерности и структуры. Для обозначения этого оператора используется комбинация символов «о.». Справа от нее находится значок примитива, к которому применяется оператор. Примеры, представленные ниже, демонстрируют работу внешнего произведения.

```
      1 2 3 4 5o.×1 2 3 4 5
1     2     3     4     5
2     4     6     8     10
3     6     9     12    15
4     8     12    16    20
5     10    15    20    25
      (⍳5)o.⌈⍳5
1     2     3     4     5
2     2     3     4     5
3     3     3     4     5
4     4     4     4     5
5     5     5     5     5
      0 1 2 3 4o.!0 1 2 3 4
1     1     1     1     1
0     1     2     3     4
0     0     1     3     6
0     0     0     1     4
0     0     0     0     1
```

Во втором примере мы использовали одноместный примитив «генератор индексов», обозначаемый греческой буквой ⍳, который в качестве своего результата выдает вектор целых чисел от 1 до N, где N — аргумент примитива.

```
      ⍳5
1 2 3 4 5
      ¯1+⍳7
0 1 2 3 4 5 6
```

Третий пример — это таблица примитива «сочетания». Те из вас, которые знакомы с высшей математикой, вероятно, узнали так называемый треугольник Паскаля, который образуется коэффициентами $(a+b)^n$. Рассмотрим некоторые применения внешнего произведения.

Как известно, многочлен степени n однозначно определяется вектором своих коэффициентов C длины $n+1$, в котором каждый C_{j+1} — это коэффициент при члене x^j. Например, вектор 4 2 1 определяет многочлен $4+2x+x^2$. Ниже приведена двухместная определяемая функция POLY, которая вычисляет значения многочлена, коэффициенты которого заданы правым аргументом, в точках, заданных значением ее левого аргумента. Мы использовали внешнее произведение для того, чтобы вычислить все члены вида x^j, и внутреннее произведение для вычисления значений многочлена.

```
      P←X POLY C
[1] P←(X∘.*⍳,¯1+⍳⍴C)+.×C
      2 POLY 4 2 1
12
      0 1 2 3 POLY 4 2 1
4 7 12 19
```

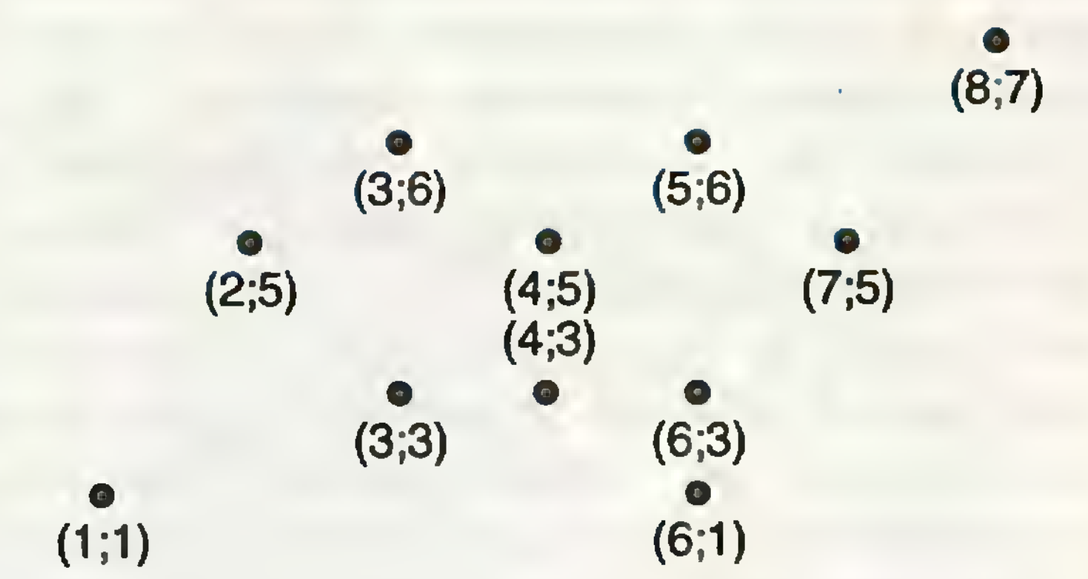
Операции дифференцирования и интегрирования многочлена также могут быть сведены к преобразованию его коэффициентов. Одноместная функция DIF и двухместная INT предназначены именно для этого. Левый аргумент INT — это значение константы интегрирования. Правые аргументы этих функций — вектор коэффициентов многочлена. Мы использовали примитив «конкатенация», обозначаемый значком «,» (запятая), который позволяет «склеить» два массива, а также примитив «отбрасывание», обозначаемый значком «↓», который отбрасывает столько элементов своего правого аргумента, сколько их задает значение левого. В нашем случае отбрасывание осуществляется от начала массива C.

```
      K←DIF C
[1] K←(⍳⍴C)×C←1↓C
      K←X⍬ INT C
[2] K←X⍬, C÷⍳⍴C
      DIF 4 2 1
2 2
      3 POLY DIF 4 2 1
8
      2 INT 4 2 3
2 4 1 1
      5 POLY 2 INT 4 2 3
172
```

Решим следующую задачу, которую мы взяли из раздела «Человек и компьютер» журнала «Наука и жизнь», где она была решена с помощью БЕЙСИКа. Пусть имеется микрорайон, состоящий из N жилых домов. На первом этаже любого дома можно устроить продуктовый магазин. Необходимо выбрать такой дом, чтобы сумма расстояний от него до всех остальных домов была минимальной. Создадим сначала матрицу ДОМА, строчки которой — координаты домов микрорайона.

```
      ДОМА←2 11⍴ 1 1 2 5 3 3 ...
      ДОМА [10;]
7     5
```

Мы уже решили задачу об определении расстояния между двумя точками.



Однако в этом случае дело обстоит сложнее, так как необходимо вычислить все возможные расстояния между всеми домами. Здесь очень удобно применить внешнее произведение. Например, все возможные разности координат X и Y домов микрорайона выражаются как

```
      X←ДОМА[;1]∘. — ДОМА[;1]
      Y←ДОМА[;2]∘.—ДОМА[;2]
```

X и Y — это матрицы. Матрица X устроена так: на пересечении *i*-й строки и *j*-го столбца находится разность координат X *i*-го и *j*-го домов. Точно так же устроена и матрица Y, но в ней хранятся разности координат Y. Обе эти матрицы симметричны, так как расстояние между домами *i* и *j* равно расстоянию между домами *j* и *i*. Матрица всевозможных квадратов расстояний вычисляется так:

```
      S←(X*2)+Y*2
```

Вектор суммы квадратов расстояний от каждого дома до остальных:

```
S←+/S
```

Элемент вектора S, находящийся на месте *i*, равен сумме квадратов расстояний от *i*-го дома до всех остальных. (На самом деле *i*-й элемент в S учитывает и расстояние от дома *i* до самого себя, но оно, естественно, равно 0.) И вот, наконец, координаты оптимального дома (или домов):

```
      ОПТИМУМ←(S=L/S)/[1]ДОМА
```

Смысл последнего выражения таков: сжать матрицу ДОМА по первому измерению, оставив только те строчки, на которых достигается минимум суммы квадратов расстояний. При решении задачи мы не стали извлекать квадратный корень из суммы квадратов, поскольку это не влияет на ее решение. Соберем теперь все дей-

ствия в одну функцию и решим задачу с нашими исходными данными.

```
      МАГАЗИН D; S
[1] S←((D[;1]°.−D[;1])*2)+(D[;2]°.−D[;2])*2
[2] 'Координаты оптимальных домов:'
[3] (S=L/S)/[1]D
      МАГАЗИН ДОМА
Координаты оптимальных домов:
4  5
```

Управление АПЛ-системой

Итак, мы провели уже достаточно продолжительный сеанс работы с АПЛ-системой. В рабочей области находится некоторое количество массивов и функций. А как можно узнать, какие именно переменные и функции нами уже определены? Как быть, если мы хотим закончить сеанс работы и продолжить его, скажем, через неделю, сохранив все те результаты, которые получены нами сегодня? Все эти вопросы относятся к компетентности АПЛ-системы. Для управления АПЛ-системой служат так называемые системные команды.

Системные команды не имеют никакого отношения к языку АПЛ и предназначены для выполнения сервисных функций в АПЛ-системе. Для того чтобы системные команды можно было отличать от выражений языка АПЛ, все они начинаются с символа «)». Никакое правильное АПЛ-выражение не может начинаться с этого символа. Если пользователь вводит с клавиатуры строчку, начинающуюся с «)», то АПЛ-система воспринимает ее не как строку языка, а как команду, обращенную непосредственно к ней. Например, если мы хотим получить на экране список всех имен переменных, существующих в рабочей области, то для этого необходимо ввести

```
)ПРМ        В        ВЕКТОР  ДОМА     М
Y           ТЕКСТ    S       ИМЕНА    X
ОПТИМУМ
```

Команда «)ПРМ» (от «переменные») распечатывает требуемый список. Аналогично команда «)ФНЦ» распечатывает список определяемых функций, существующих в настоящий момент в рабочей области:

```
)ФНЦ
СРЕДНЕЕ     РАССТОЯНИЕ  ПОИСК  INT
МАГАЗИН     ВОЗРАСТ     POLY   DIF
```

Если вы знакомы с работой БЕЙСИК-системы, то знаете, что для распечатки текста программы используется команда LIST. В АПЛ для этой цели используется команда «)ФНЦ», но уже с параметром — именем функции, текст которой мы хотели бы увидеть:

```
      )ФНЦ POLY
      P←X POLY C
[1] P←(X∘.*0, ¯1+⍳⍴C)+.×C
```

Для того чтобы сохранить всю рабочую область на диске, в АПЛ-системе имеется команда «)СОХ» (сохранить). В общем случае эта команда имеет два параметра: номер АПЛ-библиотеки и некоторое имя, под которым необходимо сохранить рабочую область в ней. По этому имени можно будет потом отыскать нашу рабочую область среди других.

```
      )СОХ 1 PRIMER
Сохранена 18-АПР-90 1 PRIMER
```

Когда команда сохранения рабочей области завершается успешно, то АПЛ-система выдает на экран сообщение, подобное этому.

После того как мы сохранили рабочую область в библиотеке, мы можем безбоязненно очистить наш рабочий лист, используя системную команду

```
      )ЧИСТ
Рабочая область свободна
      A
A
^
ОШИБКА: НЕТ ЗНАЧЕНИЯ
```

Для того чтобы вернуть всю ту обстановку в системе, которая существовала до команды очистки, достаточно выполнить команду загрузки рабочей области из АПЛ-библиотеки:

```
      )ЗАГР 1 PRIMER
Рабочая область от 18-Апр-90
      A
5
```

Эта одна команда восстанавливает значения всех массивов и тексты всех определяемых функций.

В АПЛ-системе существует довольно много команд, которые обеспечивают высокий уровень сервиса для пользователя: распечатка каталогов дисков и АПЛ-библиотек, копирование отдельных объектов, управление исполнением АПЛ-программ и многое другое.

К средствам управления АПЛ-системой относятся также специальные объекты, которые называются системными переменными и системными функциями. Мы не будем говорить о них здесь, заметим лишь, что системные функции используются для специфических действий внутри АПЛ-системы, таких, как обработка файлов, машинная графика, передача информации по сетям и т. п. Системные переменные — это массивы информации, общие для пользователя и АПЛ-системы. Изменяя их значения, пользователь может изменять некоторые параметры системы, влияющие на то, как исполняются его программы, не изменяя самих программ. Такими параметрами могут быть: точность сравнений, начало отсчета индексов в массивах, длина строки печатающего устройства, текущее значение датчика псевдослучайных чисел и т. п. Системные переменные и системные функции являются составными частями языка АПЛ.

* * *

На этом мы закончим этот раздел, который, как надеемся, дал хотя бы самое общее представление о языке программирования АПЛ. За кадром осталось подавляющее большинство его мощнейших средств, но в этом и состоит одно из достоинств языка — то, чего вы не знаете, не мешает вам в работе. Всеми его средствами не пользуется никто — АПЛ неисчерпаем. Каждый из вас наверняка сможет найти свое собственное место в этом мире и свой собственный инструментарий, наилучшим образом подходящий для решения ваших задач.

Нам пишут

Редакция продолжает получать письма по программе ПИФ, опубликованной в №5 за 1989г. Авторы этой статьи (о том, как с помощью ФОКАЛа создавать программы вычислительного и игрового характера) обобщили все предложения читателей и подготовили ответ с подробным обсуждением новых особенностей работы программы.

К сожалению, сбылись самые неприятные предчувствия авторов и в опубликованный текст программы ПИФ "вкралась" опечатка. Мы благодарим редакцию серии "Вычислительная техника и ее применение" за оперативную публикацию в №9 за 1989 год письма читателя Сергеева А., в котором приведена коррекция программы ПИФ. Справедливости ради надо отметить, что опечатка была не в четвертом столбце предпоследней строки, а в седьмом (правильное содержимое 077502). Однако в обоих вариантах исправление приводит программу ПИФ в работоспособное состояние.

Подробнее останавливаться на опечатках в примерах мы не будем, так как они имеют параллельно текст в кодах, что облегчает правильное их восприятие. Остановимся только на одной ошибке.

По вине авторов при подготовке публикации в примере на стр.33 снизу "исчезла" команда ввода символа с клавиатуры EMT 6. Первоначально данная программа в мнемоническом виде должна была выглядеть следующим образом:

```
     MOV   # A, R1   ; Поместить адрес символьной строки в R1
4$:  EMT   6         ; ввести код с клавиатуры в R0
     EMT   16        ; вывести эхо на экран
```

К сожалению, в данном случае, чтобы исправить программу, необходимо вставить команду. Это приведет к сдвигу чисел во всем примере. Однако, если в третьей строке сверху команду EMT 16 заменить на EMT 6 (числовой код

104006), то программа будет работать, правда "вслепую", т.е. без выдачи вводимого числа на экран.

По поводу замечания об останове программы, запущенной из монитора ТС. Дело в том, что этот монитор свою рабочую информацию хранит в стеке. Однако как ФОКАЛ, так и программа пользователя используют стек и для своих целей, что приводит к потере рабочей информации монитора ТС и к выходу в монитор текстов диагностики.

Сохранение рабочих ячеек отладочного монитора значительно увеличило бы программу ПИФ и сделало бы ее ориентированной только на использование с монитором ТС, что не представляется целесообразным. К тому-же, из монитора текстов всегда можно возвратиться в отладочный монитор повторной командой " + ТС".

Авторы предлагают не пользоваться этим устаревшим отладчиком. Существуют более совершенные, которые позволяют работать не только в числовых кодах, но и с использованием мнемонических обозначений машинных команд. Эти отладчики также реализуют тщательно продуманное взаимодействие с программой пользователя, исключающее подобное "вмешательство" в работу друг друга. Авторы рекомендуют в качестве такого отладчика использовать разработанную ими же программу "ГРОТ".

В процессе эксплуатации выяснились некоторые особенности исполнения команд плавающей арифметики. Мы хотели бы предложить вниманию читателей наиболее существенные их них.

1. Оказалось, что функция FSQT требует для своей работы копию сумматора в рабочих областях ФОКАЛа. Поэтому перед использованием FSQT необходимо выполнить операции FPUT @#1704.

Использование операции FREAD для перевода чисел из символьного вида в двоичный имеет один нюанс для случая отрицательного числа. Если в примере на стр.33 снизу поменять местами две команды:

```
002064 112737 MOVB #20,@#1723 ;Обязательно заполнить
002066 000020
002070 001723
002072 112304 MOVB (R3)+,R4   ;Обязательно поместить первый символ в R4
```

то после исполнения операции FREAD можно воспользоваться установленным разрядом условия N для замены знака сумматора. Тогда окончание приводимого на стр.33 примера может быть записано следующим образом:

```
002074  007075     FREAD    ;Число в сумматор
002076             BPL 5$   ;Число отрицательное?
002100  007074     FNEG     ;Да, меняем знак числа в сумматоре
002102  000000 5$:HALT
002104         A:           ;Буфер ввода
```

Собственно, операция FREAD не меняет разряд N. Он устанавливается автоматически при загрузке в регистр R4 первого байта переводимого числа, который в этом случае является минусом. Чуть ранее в статье объяснялось, что 202 код минуса в ФОКАЛе - 202. С точки зрения архитектуры ЭВМ этот байт является отрицательным числом, что приводит к установке признака N. Правда, такой-же результат будет в случае использования знака плюс (код 201) в начале строки с положительным числом, поэтому здесь следует соблюдать осторожность и придерживаться обычной практики указывания положительного числа без явно заданного знака плюс.

2. Некоторые программисты в своей работе часто используют операцию FPRINT. При этом их интересует возможность изменять значение формата выводимых чисел.

Значение формата, как известно, состоит из двух частей: первая указывает на общее количество десятичных цифр в записи выводимого числа, а вторая - количество знаков в мантиссе. В нотации ФОКАЛа это записывается, например, в виде %8.04. Перед использованием операции FPRINT в регистр R2 помещается требуемый формат, при этом старший байт содержит первую часть формата. В младший байт необходимо поместить удвоенное число позиций

(Продолжение см.стр.41)

О. П. ЛУКША

АПЛ В АНАЛИЗЕ ДАННЫХ

> От статистики не скроешься никуда.
>
> *И. Ильф и Е. Петров*
>
> Диалоговый язык АПЛ имеет восторженных сторонников среди статистиков.
>
> *Дж. Мэйндоналд*

Термин «данные» достаточно распространен. В широком смысле он означает фактический материал, являющийся основой для обсуждения или принятия решений. Однако лишь в простейших случаях данные можно использовать непосредственно. Чаще всего информация об исследуемом явлении искажена «игрой случая». Поэтому возникает потребность в приемах и методах, позволяющих вскрыть существующие закономерности и удобно представить данные. Такой аппарат представляет раздел знаний, получивший название «анализ данных» (в отечественной литературе используется также термин «прикладная статистика»).

Основная особенность, характеризующая современное состояние анализа данных, — широкое использование ЭВМ. Применению ЭВМ статистика обязана рождением принципиально новых методов — имитационное моделирование, бутстреп и других, обеспечивающих ее качественный прорыв к особенностям реальных данных, далеких от идеализированных классических схем.

Вместе с тем компьютерный фанатизм, характерный для начального этапа информационной революции, не обошел и математическую статистику. Он породил иллюзорные надежды на «...возможность построения практически сколько угодно сложных математических моделей, основанных на изящном анализе грубых экспериментальных данных» (В. В. Налимов). Эта иллюзия подкреплялась наивной верой в возможность практически полной автоматизации анализа данных, исключения из него человека.

30-летняя история безуспешных попыток реализации этой сверхзадачи привела к трезвому выводу о необходимости включения специалиста, хорошо знающего предметную область, поставляющую данные, в их анализ и управление обработкой. По образному выражению Ю. П. Адлера, анализ данных рисуется в виде «слоеного пирога», в котором «коржи» формальных операций обильно прослоены «кремом» неформальных процедур принятия решения. Отсюда ясно, что для анализа данных нужны не просто ЭВМ, а диалоговые вычислительные системы. Именно человеко-машинные системы наиболее соответствуют концепции «слоеного пирога» и являются материальной основой анализа данных.

Наиболее удачные диалоговые пакеты по математической статистике предоставляют конечному пользователю большие возможности по управлению процессом обработки. Таков, например, интегрированный пакет обработки данных STATGRAPHICS, входящий в тройку лучших на сегодняшний день систем подобного рода для персональных ЭВМ. Эффективному принятию решений способствует и последовательно реализованная в этом пакете идея графического представления данных. Вообще, если в статистике удается представить данные в форме, доступной зрительному восприятию, то возможности анализа необычайно расширяются.

Если от иллюзий полной автоматизации анализа данных мы благополучно отрешились, то еще достаточно живуче представление о возможности типизации разнородных задач и создании универсального пакета, пригодного решительно на все случаи жизни. Даже после применения такой замечательной системы, как STATGRAPHICS, исследователя часто ожидает разочарование, связанное с тем, что его задача «не вписывается» в типовую схему. Видимо, мир анализа дан-

ных скоро придет к пониманию того, что лучший пакет — это «свой пакет», скроенный под свои задачи, живущий и развивающийся вместе с представлениями исследователя о них. Это позволит избежать и соблазна бездумного применения готового пакета, об опасности чего постоянно предупреждают специалисты в области анализа данных. «Свой пакет» отражает квалификацию создателя, и это не недостаток, а благо, поскольку цена ошибок бывает слишком велика. Такая точка зрения, впрочем, не исключает заимствования готовых продуктов при ясном понимании методов, лежащих в их основе.

Практическая реализация тезиса персонализации компьютерного анализа данных и разработки программных средств его поддержки требует, конечно, и принципиально новых информационных технологий. Для этого вряд ли подойдет ФОРТРАН (скорее язык профессионалов от ЭВМ) или даже популярный ныне БЕЙСИК. Стоит, видимо, согласиться с Дж. Мэйндоналдом, утверждающим, что «...сложности проведения анализа с помощью программы, написанной на таком языке, как ФОРТРАН, часто отвлекают их (статистиков. — Авт.) внимание от статистических проблем». Альтернативой традиционным языкам программирования является язык АПЛ, предоставляющий качественно новый инструментарий при решении задач анализа данных.

Ко всем свойствам языка АПЛ, о которых уже говорилось, необходимо добавить то, что АПЛ очень хорош для решения «размыто» поставленных задач, т. е. таких, окончательная формальная постановка которых возникает лишь в процессе их решения. На практике исследователи постоянно сталкиваются с подобными задачами. Для того чтобы проиллюстрировать возможности АПЛ в анализе данных, мы расскажем почти невыдуманную историю, случившуюся в одном городе...

Регулировка движения по-научному

В городе N местные власти, обеспокоенные ростом дорожных происшествий, решили ограничить скорость движения автотранспорта и, надо отдать им должное, подошли к этому вопросу с научной точки зрения. Решено было установить зависимость между скоростью автомобиля в момент появления сигнала о возможном столкновении и длиной тормозного пути, который проходит автомобиль после включения тормозов. С этой целью были проведены испытания в различных частях города на различных типах автомашин, управляемых водителями, взятыми наугад. Результаты этих экспериментов были сведены в таблицу, фрагмент которой показан ниже. Однако анализ результатов оказался не таким простым делом, и решено было обратиться за помощью в один из консультационных центров по обработке данных.

СКОРОСТЬ (мили/ч)	ТОРМОЗНОЙ ПУТЬ (футы)
10	14
5	7
16	13
17	27
12	18
9	7
7	7
4	4
13	15
15	16
18	30
20	36
21	42
36	85
26	41
25	48
24	49
29	68
16	15
40	134
28	74
35	100
39	110
31	77

Представитель службы безопасности движения, назовем его сержант Смит, захватив с собой данные испытаний, направился в компьютерный центр, где он встретился, скажем, с доктором Джонсом.

— Я надеюсь, док, вы сможете через недельку-другую дать нам какие-нибудь рекомендации на этот счет? — произнес Смит, протягивая Джонсу ворох бумаг, обильно заполненных цифрами.

— Ну зачем же «через недельку-другую»! Если вы мне поможете, то уже сегодня вам будет о чем доложить своему начальству, сержант, — ответил Джонс.

— В полицейской академии у меня было неплохо с математикой, но, честно говоря, все эти ваши компьютеры... — Смит пошевелил пальцами около виска. — Как-то выше моего понимания.

— Успокойтесь, м-р Смит, я не буду заставлять вас программировать. Вы только поможете мне в интерпретации результатов.

— Ну, если так...

После этого оба уселись за персональный компьютер, и работа закипела.

— В обработке ваших данных, — начал программист, — нам будет помогать АПЛ-система. Первым делом мы должны ввести эту таблицу в память ЭВМ. Чтобы не запутаться, давайте определим два массива чисел СКОРОСТЬ и ПУТЬ, в которых будут храниться первый и второй столбцы нашей таблицы.

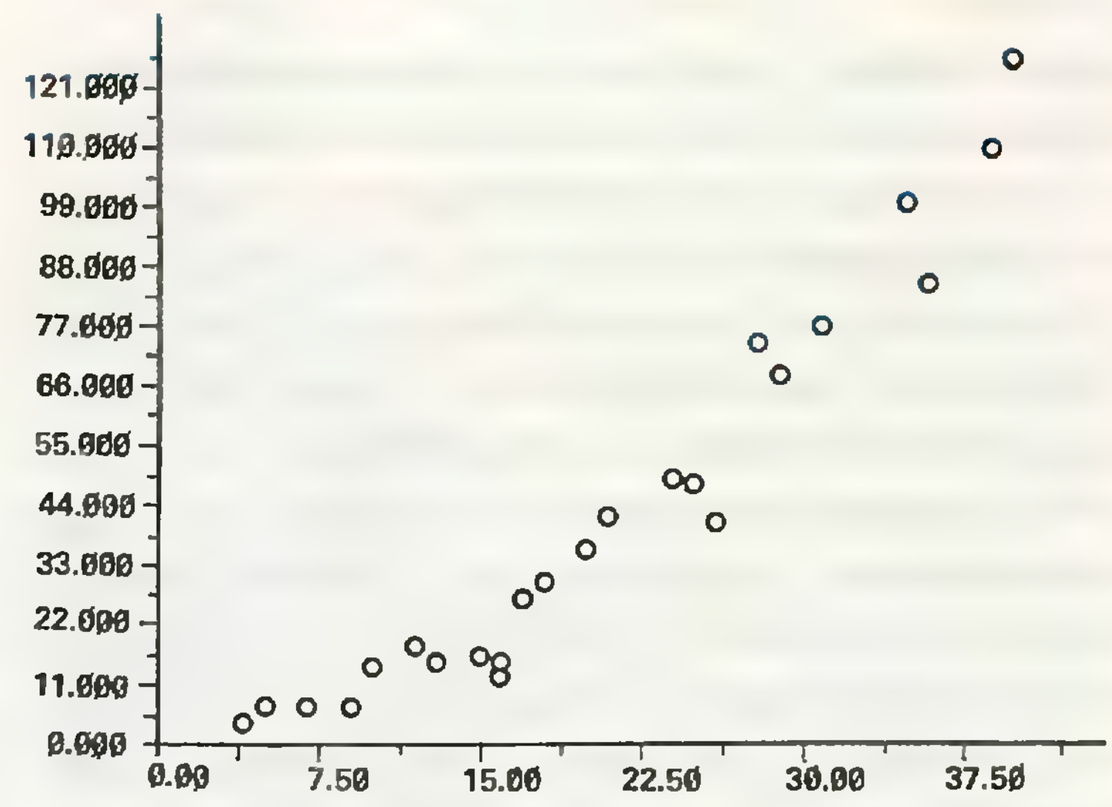
```
СКОРОСТЬ←10 5 16 17 12 9 7 4 13 15 18 20 21
36 26...
ПУТЬ←14 7 13 27 18 7 7 4 15 16 30 36 42 85 41
48...
```

— И как же теперь получить зависимость между этими числами? — сказал Смит.

— Существуют различные формы представления зависимостей. Проще всего построить точечную диаграмму. Для этого у нас есть программа ТОЧКИ, — ответил Джонс, нажимая на клавиши.

```
ТОЧКИ ПУТЬ ОТ СКОРОСТЬ
```

— Действительно, такое представление более наглядно, — сказал Джонс, разглядывая рисунок на экране дисплея. — Видно, что тормозной путь в принципе растет с увеличением скорости. Но зависимость между ними слишком смутна.



— Второй способ заключается в преобразовании исходной таблицы к более удобному для восприятия виду. Давайте упорядочим ее, например, по возрастанию значений скорости. В этом нам поможет встроенная функция АПЛ, обозначаемая значком «△».

```
      △СКОРОСТЬ
8 2 7 6 1 5 9 10 3 19 4 11 12 13 17 16 15 21 18 24
22 14 23 20
```

— Ваша машина ошибается, это не скорости автомобилей!

— Вы совершенно правы, сержант, это не сами скорости, а лишь номера автомобилей от 1 до 24 в порядке увеличения их скоростей.

— По-моему, это лишняя информация, — недовольно сказал полицейский. — Ведь нас же интересуют скорости!

— Не будем торопиться с выводами относительно «лишней» информации, — возразил программист. — Ну а скорости мы легко можем получить, индексируя массив СКОРОСТЬ полученными числами. Например, скорости трех самых медленных автомобилей равны...

```
      СКОРОСТЬ [8 2 7]
4 5 7
```

— Индексирование, — пояснил Джонс, — это такая операция, которая позволяет извлечь из массива интересующие нас элементы, указав их номера в квадратных скобках, следу-

ющих непосредственно за именем массива. Таким образом, мы сейчас «вытащили» из массива СКОРОСТЬ три элемента с номерами 8, 2 и 7.

— Теперь понимаю! — воскликнул Смит. — Значит, весь массив скоростей, отсортированных по возрастанию... Дайте-ка я попробую сам!

```
      СКОРОСТЬ [⍋ СКОРОСТЬ]
4 5 7 9 10 12 13 15 16 17 18 20 21 24 25 26 28 29
31 35 36 39 40
```

— Вы делаете успехи, м-р Смит, — похвалил Джонс.

Сержант, явно польщенный похвалой профессионала, собрался ввести еще одну строчку: — Очевидно, то же самое надо сделать и с массивом ПУТЬ?

— Не спешите, — предупредил Джонс. — Ведь мы хотим сохранить соответствие между скоростью и тормозным путем. Но если просто отсортировать массив ПУТЬ по возрастанию, то зависимость между ними может потеряться.

— Постойте, постойте! Конечно! Надо изменить порядок элементов в массиве ПУТЬ так же, как мы изменили его, упорядочив массив СКОРОСТЬ, т. е.

```
      ПУТЬ [⍋СКОРОСТЬ]
4 7 7 7 14 18 15 16 13 15 27 30 36 42 49 48 41 74
68 77 100 85 110 134
```

— Вот и пригодилась «лишняя» информация... — съязвил Джонс.

— Согласен. А можно ли теперь увидеть всю таблицу целиком?

— Это несложно сделать с помощью программы И. Однако сначала давайте запомним в рабочей области АПЛ-системы отсортированные массивы, поскольку они нам еще понадобятся.

```
      СКОРОСТЬ1 ← СКОРОСТЬ [⍋СКОРОСТЬ]
      ПУТЬ1→ПУТЬ [⍋СКОРОСТЬ]
```

— Ну а вот и наша таблица...

```
      (⍳24) СКОРОСТЬ1 И ПУТЬ1
1      4      4
2      5      7
3      7      7
4      9      7
5     10     14
6     12     18
7     13     15
8     15     16
9     16     13
10    16     15
11    17     27
12    18     30
13    20     36
14    21     42
15    24     49
16    25     48
17    26     41
18    28     74
19    29     68
20    31     77
21    35    100
22    36     85
23    39    110
24    40    134
```

— А что это за «⍳24»? — недоумевающе спросил Смит.

— Греческой буквой «йота» в языке АПЛ обозначается примитив «генератор индексов». В качестве результата он выдает массив целых чисел от 1 до числа, указанного справа от значка «⍳». Я сделал это, чтобы пронумеровать строки в таблице.

— Такая таблица, конечно, удобнее исходной, — произнес сержант, задумчиво рассматривая изображение на экране. — Но я бы не сказал, что она приблизила нас к цели. Скорее наоборот! Посмотрите-ка, м-р Джонс, у автомобиля, имеющего скорость 18 миль в час, тормозной путь больше, чем у автомобиля, имеющего скорость 17, а у автомобиля со скоростью 29 он меньше, чем у автомобиля со скоростью 28?!

— Я думаю, что на это были свои причины, — спокойно ответил Джонс. — Существует масса случайных факторов, которые могли привести к такому несоответствию. Автомобили могли иметь различную массу. У них могли быть тормоза, отличающиеся по качеству. Водители ведь тоже разные: одни моментально реагировали на сигнал, а другие чуть позже. А состояние дороги? Одним словом, всех причин не перечесть...

— Боюсь, мое начальство не удовлетворят рассуждения о случайностях... — хмуро процедил сержант.

— Для уменьшения влияния слу-

чайных факторов можно применить прием группировки. Разобьем, например, массив скоростей на 3 группы по 8 чисел в каждой, а затем вычислим средние значения скорости и тормозного пути для каждой из этих групп. Процедура вычислений для первой группы такова:

```
  (+/СКОРОСТЬ1[1 2 3 4 5 6 7 8])÷8
9.375
  (+/ПУТЬ1[1 2 3 4 5 6 7 8])÷8
11
```

— На самом деле, мы могли бы не перечислять явно все восемь номеров, — продолжал Джонс, — а воспользоваться генератором индексов. Давайте подсчитаем и запомним все нужные нам средние значения.

```
С1←(+/СКОРОСТЬ1[⍳8])÷8
П1←(+/ПУТЬ1[⍳8])÷8
С2←(+/СКОРОСТЬ1[8+⍳8])÷8
П2←(+/ПУТЬ1[8+8])÷8
С3←(+/СКОРОСТЬ1[16+⍳8])÷8
П3←(+/ПУТЬ1[16+⍳8])÷8
```

— Теперь мы можем построить сводную таблицу результатов, — сказал программист, закончив работу.

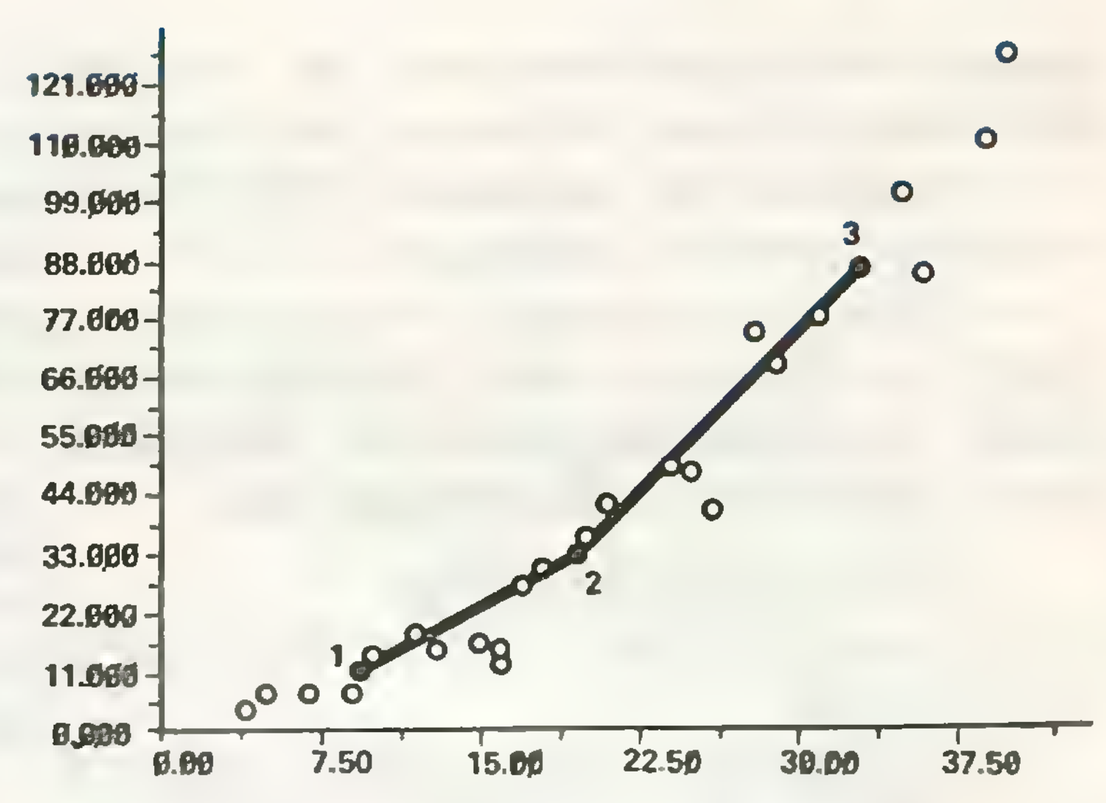
```
  (С1, С2, С3) И П1, П2, П3
9.375     11
19.625    32.5
33        86.125
```

— А если нас интересует средний тормозной путь автомобиля, имеющего скорость 11? — спросил Смит. — Как его подсчитать?

— Простейшее решение этой задачи — графическое, — ответил Джонс. — Нужно соединить прямыми линиями три точки, координаты которых указаны в строках полученной нами таблицы. Это легко можно сделать с помощью программы ЛИНИЯ.

```
  ЛИНИЯ (П1, П2, П3) ОТ С1, С2, С3
```

— Эта ломаная линия показывает уже непрерывную зависимость между скоростью автомобиля и его тормозным путем. Да и тех противоречивых результатов, что были раньше, здесь уже нет, — закончил программист.



— Пользоваться графиком не всегда удобно, — возразил полицейский.

— Согласен. В ряде случаев лучше иметь некоторое математическое выражение, позволяющее вычислять значение тормозного пути по заданной скорости автомобиля. Каждый из отрезков, которыми мы соединили найденные точки, описывается уравнением вида $y=a+bx$ с разными значениями параметров a и b.

— Как же рассчитать a и b? — спросил Смит.

— Нахождение значений a и b для прямой, соединяющей точки 1 и 2 на нашем графике, сводится к решению системы двух линейных уравнений с двумя неизвестными:

$$a+9.375\times b=11$$
$$a+9.625\times b=32.5$$

— Эта система, — продолжил Джонс, — отражает тот очевидный факт, что прямая проходит через точки с координатами [9.375;11] и [19.625;32.5].

— Да, действительно, — неуверенно сказал Смит. — Что-то смутно припоминаю. Кажется, в академии мы это проходили. Но чем же здесь нам может помочь АПЛ-система?

— Для этого у нас есть функция, которая обозначается значком «деление в прямоугольнике»⌹

— Похоже на костяшку домино...

— Ее иногда так и называют, — согласился Джонс. — Однако на самом деле эта функция называется матричным делением. Для того чтобы воспользоваться ею для решения нашей системы уравнений, необходимо составить матрицу коэффициен-

тов при неизвестных *a* и *b*, а также вектор правых частей системы...

— Здесь мне непонятны две вещи, — прервал программиста Смит. — Во-первых, что такое матрица? А во-вторых, если коэффициенты при *b* — это 9.375 и 19.625, то при *a* никаких коэффициентов нет!

— Нет ничего проще! — ответил Джонс. — Матрица — это всего-навсего таблица чисел, подобная той, с которой мы уже имели дело при представлении экспериментальных данных. Ну а коэффициенты при неизвестной величине *a* — единицы. Таким образом, матрица коэффициентов выглядит так:

```
1   9.375
1  19.625
```

— А вектор свободных членов вот так:

```
11  32.5
```

— Теперь мы сформируем и запомним матрицу коэффициентов в рабочей области АПЛ-системы, — говорит Джонс, набирая на клавиатуре очередную строчку.

```
  КОЭФ1←2 2⍴ 1 9.375 1 19.625
  КОЭФ1
1      9.375
1     19.625
```

— По-моему, я понял! — воскликнул Смит. — Значок «⍴» — это, вероятно, функция АПЛ, предназначенная для формирования матриц? Слева от него вы указали размеры матрицы, а справа — просто перечислили все ее элементы!

— Вы совершенно правы, сержант, — похвалил Джонс. — Эту функцию, она, кстати, называется «перестройка», можно использовать не только для формирования матриц, но и массивов произвольных размерностей. А как вы думаете, что я делаю сейчас?

```
    11  32.5 ⌹ коэф1
8.6646    2.0976
```

— Это и есть *a* и *b*?

— Да. Это параметры отрезка, соединяющего точки 1 и 2. А вот значения *a* и *b* для второго отрезка:

```
  32.5  86.125 ⌹ 2 2⍴1  19.625  1  33
46.183  4.0093
```

— Теперь мы напишем математическое выражение, с помощью которого можно вычислить значение тормозного пути *y* по заданной скорости автомобиля *x*, — сказал программист.

$y=8.6646+2.0976x$ для $9.375\leqslant x\leqslant 19.625$
$y=-46.183+4.0093x$ для $19.625\leqslant x\leqslant 33$

— А можно как-то истолковать вычисленные нами значения *a* и *b*? — спросил полицейский.

— Вы задали очень важный вопрос. Соотношения, подобные тем, что мы получили, обычно называют моделями. Математические модели, правильно описывающие объективные закономерности исследуемого явления, как правило, содержат параметры, которые имеют хорошо интерпретируемый физический смысл. Для полученных зависимостей параметр *a* сам по себе не имеет смыслового значения, представляя собой лишь начало отсчета. Действительно, если мы продолжим первый отрезок до его пересечения с осью расстояний, то окажется, что при нулевой скорости автомобиля величина тормозного пути будет равна отрицательной величине?! С другой стороны, параметр *b* имеет ясный физический смысл. Он показывает, на сколько в среднем изменяется длина тормозного пути при изменении величины скорости на единицу. Из полученных нами результатов следует, что скорость изменения тормозного пути между точками 2 и 3 значительно выше, чем между точками 1 и 2. Это наводит на мысль, что искомая зависимость имеет более сложный вид...

— А что если провести какую-нибудь кривую, но уже через все 3 точки сразу? — предложил Смит.

— Я и сам склоняюсь к этой мысли... — согласился Джонс. — Можно, например, использовать квадратичную зависимость вида $y=a+bx+cx^2$.

— Ага, — сказал Смит. — Три точки, три неизвестных параметра *a*,*b* и *c*... Значит, нужно решать систему из трех уравнений с тремя неизвестными?

— Да, — подтвердил программист.

— И на АПЛ это ничуть не сложнее, чем решение системы из двух уравнений. Надо только правильно составить матрицу коэффициентов...

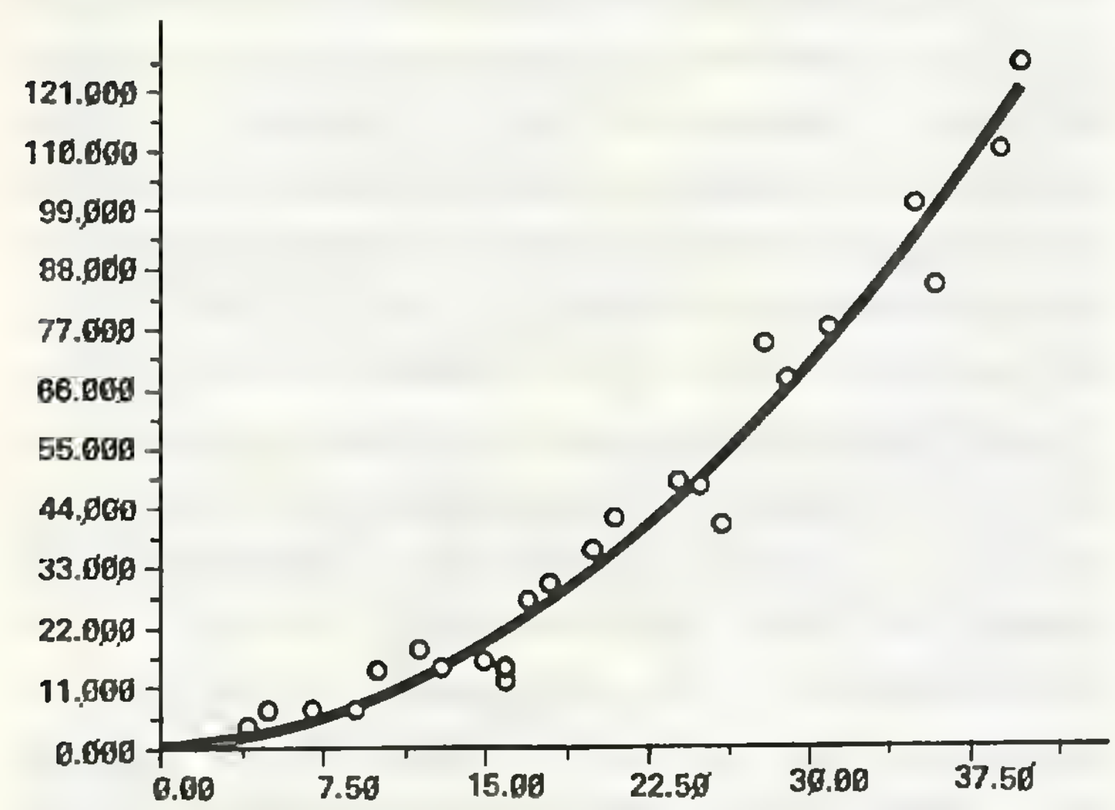
```
      КОЭФ2←1 1 1 И (С1,С2,С3) И (С1,С2,С3)*2
      КОЭФ2
1       9.375     87.891
1      19.625    385.14
1      33        1089
АВС ← 11 32.5 86. 125 ⌹ КОЭФ 2
    АВС
6.2238 ¯0.24918 0.8092
```

— Ну вот, вектор АВС — это и есть три искомых параметра.

— Хорошо бы теперь увидеть эту кривую... — попросил Смит.

— Пожалуйста...

ЛИНИЯ (СКОРОСТЬ POLY АВС) ОТ СКОРОСТЬ



— Смотрите, док! Опять получается, что при нулевой скорости тормозной путь отличен от 0... — разочарованно произнес полицейский.

— Вы правы... А что если мы попытаемся приблизить наши данные с помощью кривой вида $y=bx+cx^2$. Эта кривая уже точно пройдет через начало координат...

— Ну, это для нас пара пустяков, — оживился Смит. — Надо решить систему. Постойте, постойте! Здесь что-то не так! Ведь точек три, следовательно, и уравнений будет три, а неизвестных параметров только два. Это означает, что мы должны решить систему трех уравнений с двумя неизвестными. Это мы точно не проходили! Ведь любая пара из этих трех уравнений дает решение?

— Действительно, — ответил программист. — В общем случае, нельзя провести кривую, которая описывается уравнением с двумя параметрами через три произвольные точки, но это можно сделать приближенно. Например, среди всех возможных кривых данного вида найти такую, которая проходит на минимальном суммарном расстоянии от всех точек.

— Но это же очень сложная задача!

— Для пользователя АПЛ-системы эта задача ничуть не сложнее тех, что мы уже решали. Для начала составим матрицу коэффициентов системы:

```
      КОЭФ3←КОЭФ2[;2 3]
      КОЭФ3
9.375     87.891
19.625   385.14
33       1089
```

— Понимаю, — сказал Смит. — Вы взяли из предыдущей матрицы второй и третий столбцы. Конечно, ведь неизвестных только два! А дальше?

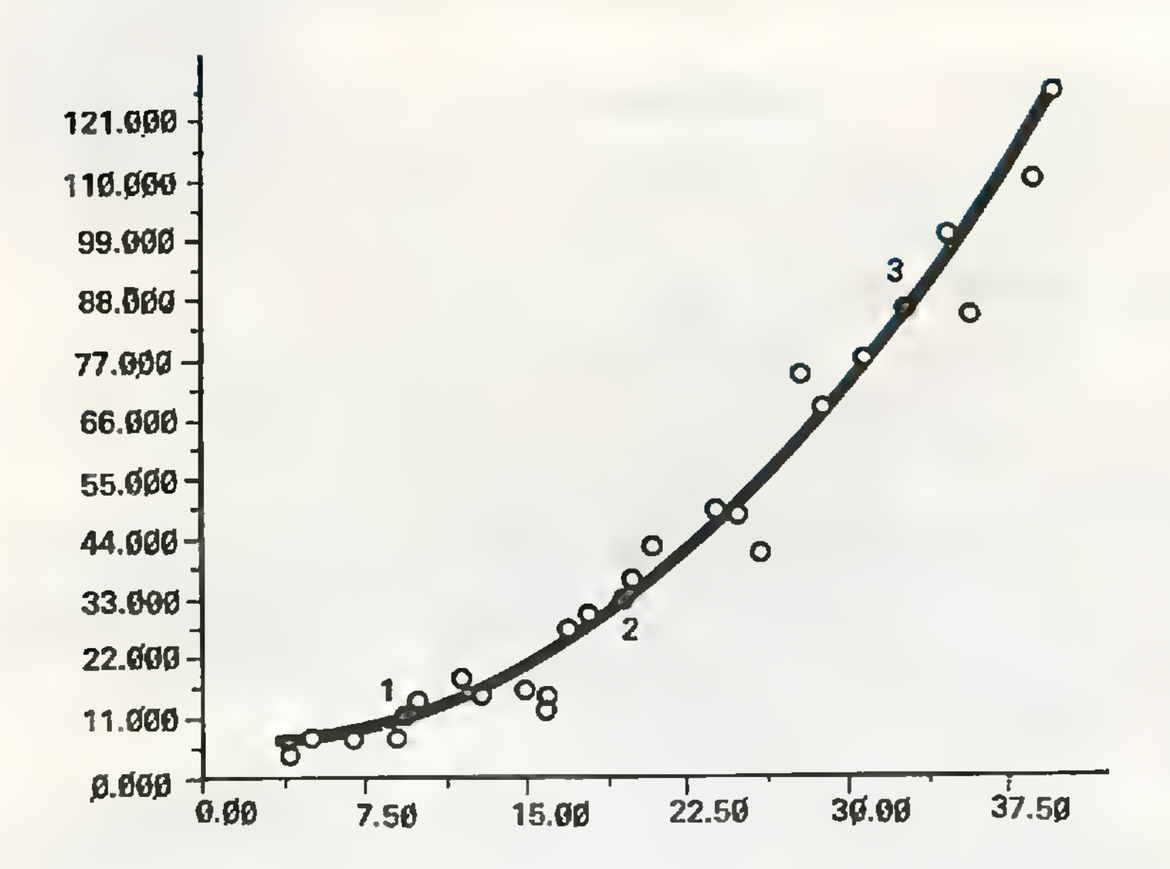
```
      ВС ← 11 32.5 86.125 ⌹ КОЭФ 3
    ВС
0.40968  0.06641
```

— Это и есть искомые параметры b и c, — объявил Джонс. — В тех случаях, когда количество уравнений системы превышает число неизвестных, функция «домино» находит приближенное решение. При этом гарантируется, что сумма квадратов отклонений экспериментальных точек от кривой, описываемой уравнением с найденными значениями параметров, — минимальна. Этот прием в математике называют методом наименьших квадратов. А вот и график этой кривой:

ЛИНИЯ (СКОРОСТЬ POLY ВС)ОТ СКОРОСТЬ

— А аналитическая формула, — закончил Джонс, — выглядит так: $y=0.40968x+0.06641x^2$.

— Эврика! — воскликнул Смит. — Я, кажется, понял смысл составляющей bx! Это та часть пути, которую проходит автомобиль от подачи сигнала до включения тормозов. Ведь водители не обладают мгновенной реакцией. Таким образом, 0.40968 —



это среднее время реакции водителей.

— Пожалуй... Только с маленькой оговоркой. Надо учесть разную размерность *x* и *y*, но это уже задача нехитрая. Сложнее обстоит дело со второй составляющей тормозного пути — cx^2. Мы предположили, что величина тормозного пути, который проходит автомобиль после начала торможения, имеет квадратичную зависимость от его скорости, но это только гипотеза. Возможно, эта зависимость более сложная, скажем, кубическая?

— Знаете, м-р Джонс, я все-таки думаю, что квадратичная зависимость хорошо интерпретирует наши данные. Если «прокрутить» эксперимент с автомобилем наоборот, то это будет очень похоже на опыт со свободно падающим телом, а ведь его движение также описывается квадратным уравнением. Кстати, в этом случае параметр *c* тесно связан с ускорением торможения, которое зависит от качества тормозов и состояния дороги!

— Ваши доводы достаточно убедительны, м-р Смит, — согласился программист.

— И еще, мне в голову пришла интересная мысль, — сказал полицейский. — В результате обработки экспериментальных данных мы получили характеристики, которые непосредственно в опыте не контролировались! Среднее время реакции, например. Это просто удивительно. Скажите, м-р Джонс, а можно было бы целиком поручить машине поиск зависимости пути от скорости?

— Конечно, — ответил Джонс. — Можно было бы предложить некоторую формальную процедуру подбора интересующей нас зависимости. Однако ЭВМ никогда не сможет полностью заменить человека. Ее следует рассматривать не более чем партнера, облегчающего процесс поиска истины. Во всяком случае, устраняться от решения такой задачи, как наша, я бы не стал. Мне кажется, что теперь, когда мы почти решили ее, вы и сами это понимаете.

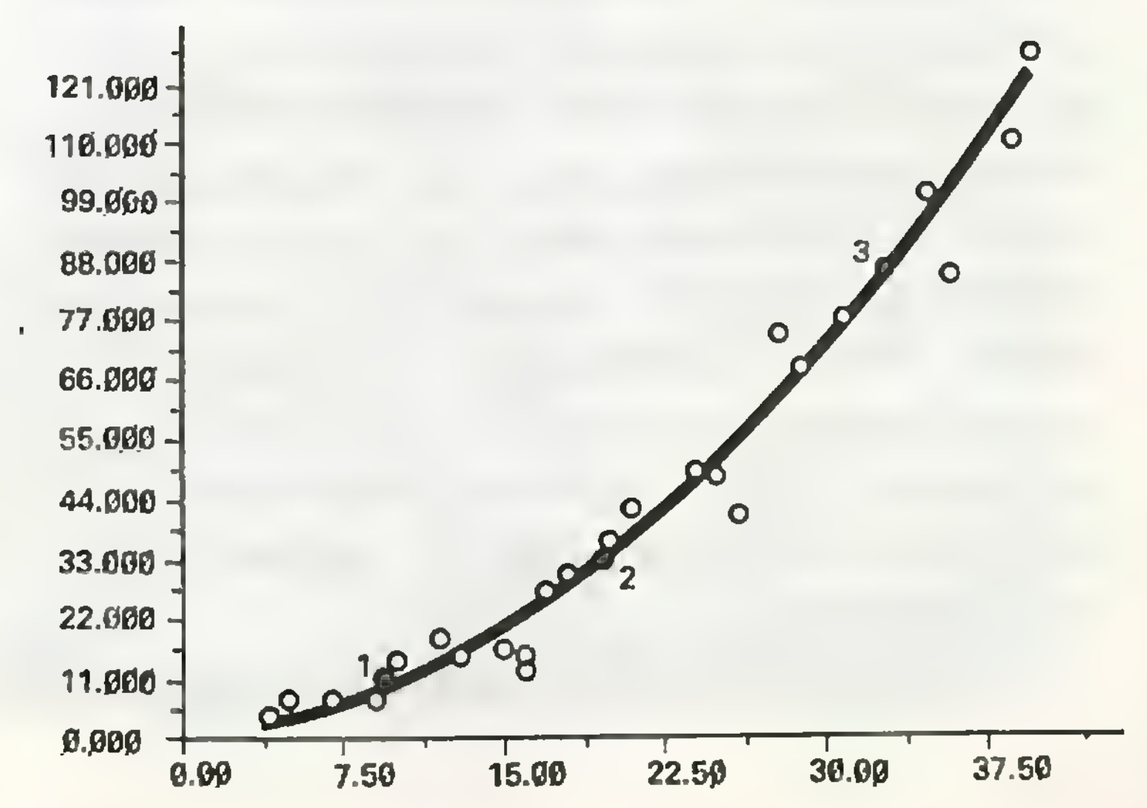
— Почти? — удивился Смит. — Что же еще?

— Мы выполнили наиболее трудный, неформальный этап — подобрали структуру модели с достаточно ясным физическим смыслом. Остается сравнительно трудоемкая, но хорошо мне понятная часть работы, связанная с уточнением параметров модели, анализом ее точности и т. п. Короче, необходимо провести целый комплекс исследований, называемый регрессионным анализом.

— Что значит уточнить значения параметров? — спросил Смит.

— При расчете значений параметров *b* и *c* нашей модели мы использовали лишь три средние точки, полученные после группировки данных эксперимента. Теперь ясно, что этот этап можно было не делать, а сразу провести кривую $y=bx+cx$ так, чтобы она прошла на минимальном расстоянии от всех экспериментальных точек. Вот так:

```
МОДЕЛЬ ← ПУТЬ [+] СКОРОСТЬ
И СКОРОСТЬ * 2
МОДЕЛЬ
```



```
МОДЕЛЬ ← ПУТЬ ⌹ СКОРОСТЬ И СКОРОСТЬ
*2 МОДЕЛЬ
0.27112      0.06966
      ЛИНИЯ (СКОРОСТЬ POLY МОДЕЛЬ) ОТ
      СКОРОСТЬ
```

— А окончательно оформленные результаты вы получите завтра, — закончил Джонс.

— Жаль, что мы не додумались до этого сразу... — вздохнул полицейский. — Хотя тогда я бы никогда не узнал, что такое АПЛ! Никогда не предполагал, что анализ данных такое увлекательное занятие. Вероятно, это как-то связано с тем, что АПЛ — специализированное средство для этого?

— Ни в коем случае, — ответил программист. — Вы увидели лишь ничтожную часть из всего спектра возможностей АПЛ. Существуют не менее интересные сферы его применения в экономике, моделировании, обучении — всего не перечесть. АПЛ — универсальный язык.

— Надо будет посоветовать моему руководству приобрести АПЛ-компьютер. Мне кажется, что теперь я и сам могу кое-что сделать даже без вашей помощи...

Чем же АПЛ привлекает статистиков?

Итак, подведем некоторые итоги, которые проясняют причины восторженности статистиков по отношению к АПЛ.

1. Язык АПЛ — это фактически язык статистики. Он включает в себя в качестве подмножества все операционные средства матричной алгебры, которая буквально пронизывает современный математический аппарат анализа данных. Алгоритм, записанный в матричной форме, можно «с листа» проиграть на АПЛ-системе. Но самое удивительное то, что операционный состав языка АПЛ гораздо шире, что позволяет лаконично и точно выразить любой статистический алгоритм. АПЛ обеспечивает статистикам полную свободу делать все, что захочешь.

2. АПЛ особенно хорош для решения «размыто» поставленных задач анализа данных, когда в процессе решения уточняются и цели исследования, и необходимые для этого средства.

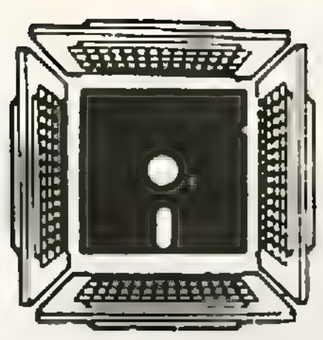
3. АПЛ — инструмент конструирования и апробации алгоритмов анализа. На АПЛ без труда порождаются данные требуемой структуры, на которых можно проводить испытания своих и чужих самых свежих идей — статистические эксперименты.

4. Профессиональные программисты в союзе со специалистами в области анализа данных могут на порядок увеличить производительность труда при создании коммерческих пакетов по статистическому анализу. Упоминавшийся выше пакет STATGRAPHICS также написан на АПЛ. По прогнозам экспертов, доля статистических АПЛ-пакетов будет неуклонно возрастать.

5. АПЛ — идеальный помощник в беседе специалиста по анализу данных с заказчиком программного статистического приложения. Возможно создание макета или даже готового программного продукта с заказчиком «за спиной».

6. Если вы — преподаватель, специализирующийся на статистике, то АПЛ позволит придать учебному процессу новое качество. Студент пойдет с конспектом ваших лекций прямо к ЭВМ. Вы сможете без труда подготовить лабораторный практикум, не обращаясь за помощью к профессионалам из вычислительного центра, которые постоянно заняты своими собственными делами и не склонны вникать в ваши проблемы.

7. Если вы хотите всерьез применять методы анализа данных в своей профессиональной деятельности, но не знаете ни статистики, ни программирования, используйте АПЛ-систему. Вы значительно сократите время изучения того и другого в отдельности, овладевая АПЛ и статистикой сразу. Подходят любые методические принципы изучения: «АПЛ через статистику» или «Статистика через АПЛ».



А. О. СКОМОРОХОВ

ВВЕДЕНИЕ В ЭКСПЕРТНЫЕ СИСТЕМЫ НА АПЛ

НЕОКОНЧЕННАЯ ПЬЕСА ДЛЯ РУССКОГО РАЗГОВОРНОГО И АПЛ АЛГОРИТМИЧЕСКОГО ЯЗЫКОВ

Действующие лица:

Автор — придумал все то, что говорят Читатель, Эксперт и Компьютер.

Читатель — обычный инженер, помешан на компьютерах, женат, имеет детей, зарплата 175, не был, не привлекался, не участвовал.

Эксперт — дядя Коля, бывший бухгалтер, а ныне — капитан сборной двора по «забиванию козла» и активный член микрорайонной лиги «Пенсионеры за перестройку».

Компьютер — говорит только тогда, когда его просят.

Сцена первая

Действие происходит в обычной малогабаритной квартире. В комнате трое: Автор, Читатель и Компьютер. Только что закончилась трансляция заседания Верховного Совета. Ночь. В голову лезут странные мысли.

Читатель (ни к кому не обращаясь). Интересно, а можно ли было заранее сказать, чем закончится голосование по последнему законопроекту?

Автор. Думаю, что для этой цели можно воспользоваться вашим компьютером. (Оба смотрят на Компьютер, но тот безмолвствует.)

Читатель. Сначала объясните, что должен выдавать Компьютер в качестве решения задачи?

Автор. Вообще говоря, возможны три исхода при решении того или иного вопроса Верховным Советом: решение положительно (закон, постановление, назначение и т. п. — принимается), решение отрицательно или рассмотрение переносится на более поздний срок. Один из этих исходов и должен предсказать компьютер.

Читатель (с иронией). Хотелось бы получить формулы, необходимые для расчетов.

Автор. Оценил ваш юмор, но таких формул, естественно, нет. Однако ситуация не безнадежна. И эту, и множество других, еще более сложных задач, в состоянии удовлетворительно решать человек.

Читатель. Ну я, например, не в состоянии.

Автор. Конечно. Речь идет о человеке, который является специалистом в данной области и обладает необходимыми знаниями и опытом. Таких людей обычно называют экспертами. Применительно к нашей задаче, экспертами могут быть, например, журналист, длительное время освещающий работу парламента, опытный депутат или хотя бы ваш сосед дядя Коля, который, насколько мне известно, смотрит все телевизионные репортажи и даже конспектирует отдельные выступления.

Читатель. А причем здесь компьютер?

Автор. Задачей теории и практики экспертных систем как раз и является создание «электронных экспертов» — передача ЭВМ знаний и опыта экспертов, а также их умения делать выводы.

Читатель. Но по рассматриваемому вопросу у меня нет таких знаний и опыта! Единственно, что я могу предложить, — это взять три бумажки, написать на одной из них «ПРИНЯТО», на другой — «ОТКЛОНЕНО», а на третьей — «ПЕРЕНЕСЕНО». Потом положить эти бумажки в шапку, перемешать и вытащить результат «предсказания».

Автор. Для начала совсем неплохо. Кстати, гадалки на картах уже давно применяют подобные экспертные системы, и многих клиентов это вполне устраивает. Однако давайте запрограммируем вашу шапку на компьютере. Я рекомендую использовать для этой цели АПЛ.

Читатель. Я разбираюсь в АПЛ не больше, чем в экспертных системах...

Автор. Бросьте! В АПЛ разбираются все, просто не все об этом знают. Смотрите! (Набирает строчку на клавиатуре.)

```
  2+2
4
```

Читатель. Ну, так-то и я могу...

Автор. А я о чем... Продолжаем. Сейчас я организую символьную матрицу из трех строк, каждая из которых — это одно из слов, обозначающих исход голосования.

```
  ИСХОД←3 10⍴ 'ПРИНЯТО ОТКЛОНЕНО
ПЕРЕНЕСЕНО'
  ИСХОД
ПРИНЯТО
ОТКЛОНЕНО
ПЕРЕНЕСЕНО
  ИСХОД [2;]
ОТКЛОНЕНО
```

Автор. Здесь я попросил Компьютер распечатать строчку с номером 2, что он с успехом и выполнил.

Читатель. Вот тут я не пойму. Ведь когда я тащу бумажку из шапки, мне неизвестно, что на ней будет написано. А вы, попросив Компьютер распечатать вторую строку матрицы, заранее знали, каким будет ответ.

Автор. Совершенно верно. Для того чтобы наша программа была похожа на настоящую шапку, мы должны внести в ее работу элемент случайности. Это позволяет сделать примитив языка АПЛ, который называется «случайное число» и обозначается значком «?». «Случайное

число» — это одноместный скалярный примитив. Его аргументом должно быть целое число. Если N — аргумент, то результатом ?N является также целое число, случайным образом выбранное из множества целых чисел 1, 2, ..., N. При этом все эти числа имеют одинаковую вероятность быть выбранными. Например, бросание монеты на АПЛ можно смоделировать так:

```
  ?2
1
  ?2
1
  ?2
2
```

где 1 и 2 обозначают «орла» и «решку». Поскольку «?» — скалярный примитив, то он может применяться и к векторам. Например, модель бросания двух кубиков при игре «в кости» выглядит так:

```
  ?6 6
3 5
  ?6 6
1 6
```

Теперь мы можем смоделировать работу шапки следующим образом.

```
  ИСХОД [?3;]
ПЕРЕНЕСЕНО
  ИСХОД [?3;]
ПРИНЯТО
```

Читатель (с блеском в глазах). Очень интересно? Завтра, при утверждении состава кабинета министров я обязательно испытаю нашу экспертную систему.

Автор. Желаю успеха! Да, и вот еще что. Я тут книжечку оставляю. Л. Гилман и А. Роуз. «Курс АПЛ. Диалоговый подход». — М.: Мир, 1979 — лучше нее книги по АПЛ вам не найти.

Сцена вторая

Несколько дней спустя. По телевизору — очередная трансляция заседания Верховного Совета. В комнате Читатель и Компьютер. Все последние вечера Читатель проводит у телевизора, пытаясь предсказывать результаты голосований. Совпадения, к сожалению, очень редки. Игра в предсказания начинает надоедать. В комнату входят Автор и Эксперт.

Автор (Эксперту). Вот это тот самый Читатель, который собирается заменить вас компьютером. Однако он, кажется, чем-то недоволен.

Читатель. Здравствуйте, дядя Коля! (рассказывает Эксперту о своих неудачах.)

Эксперт. Ваша ошибка в том, что вы считаете все исходы равновероятными. Я долгое время наблюдаю за работой Верховного Совета. В 50% случаев внесенный на рассмотрение вопрос после обсуждения откладывался. Только в 30% случаев по рассматриваемому вопросу сразу принималось положительное решение, а в 20% случаев — отрицательное. Вы просто не учитываете нереш... гм... взвешенности нашего парламента, его стремление не решать вопросы с ходу.

Читатель (обдумывая какую-то идею). Понятно. Надо положить в «шапку» 10 бумажек. На пяти из них написать «ПЕРЕНЕСЕНО», на трех — «ПРИНЯТО» и на двух — «ОТКЛОНЕНО». Таким образом, наша процедура принятия решения на АПЛ будет выглядеть так:

```
  ИСХОД [1 1 1 2 2 3 3 3 3 3 [?10];]
ПЕРЕНЕСЕНО
  ИСХОД [1 1 1 2 2 3 3 3 3 3 [?10];]
ПРИНЯТО
  ИСХОД [1 1 1 2 2 3 3 3 3 3 [?10];]
ПЕРЕНЕСЕНО
```

Автор. Поясните, пожалуйста.

Читатель. Вместо того чтобы нарезать новые бумажки со словами, т. е. увеличивать число строк в матрице ИСХОД, я использовал числовой вектор из 10 элементов, который содержит 30% единиц, 20% двоек и 50% троек. Затем я индексирую этот вектор случайным числом, выбранным из множества целых чисел от 1 до 10, а полученное в результате этого число использую для индексирования матрицы ИСХОД. Вы обратили внимание на то, что теперь Компьютер чаще будет предсказывать перенос вопросов?

Автор. Обратил, обратил. Но, во-первых, ваша запись неудачна. Предположим, завтра выяснится, что реальное соотношение вероятностей исходов (распределение вероятности) составляет 31, 47 и 22%. Вам тогда придется полностью изменить вашу программу! А во-вторых, ваша экспертная система станет, конечно, чуть-чуть «умнее», но до Эксперта ей все еще далеко.

Читатель. Что касается вашего первого замечания, то я его полностью принимаю и сегодня же напишу определяемую функцию АПЛ, которая будет способна работать с любым распределением вероятности. Ну а по поводу второго нам лучше поговорить после испытаний нового варианта системы.

Сцена третья

В той же комнате на следующий день. Автор, Читатель и Эксперт — за компьютером.

Читатель (автору). Вот какая функция у меня получилась:

```
  ГОЛОСОВАНИЕ
[1] ИСХОД [1 ++/ (+ \ PH) < (?30000) ÷ 30000;]

  PH
0.3 0.2 0.5
  ГОЛОСОВАНИЕ
ПЕРЕНЕСЕНО
  ГОЛОСОВАНИЕ
ПЕРЕНЕСЕНО
  ГОЛОСОВАНИЕ
ОТКЛОНЕНО
```

Вектор PH хранит вероятности гипотез «ПРИНЯТО», «ОТКЛОНЕНО» и «ПЕРЕНЕСЕНО». Теперь достаточно изменить его содержимое при изменении вероятностей выбора исходов. Функция ГОЛОСОВАНИЕ работает следующим образом. Сначала я получаю случайное число в диапазоне от 1 до 30 000, которое делю на 30 000. Так я моделирую случайную величину, равномерно распределенную между 0 и 1. Затем с помощью оператора «сканирование», применяемого к сло-

жению, вычисляю границы интервалов, длины которых соотносятся так, как и вероятности исходов. Сканирование напоминает редукцию, но если редукция вычисляет только сумму вектора, то сканирование выдает и все промежуточные результаты вычислений. Продемонстрирую это на примере:

```
  PH
0.3 0.2 0.5
  +/PH
1
  +\PH
0.3 0.5 1
```

Затем я выясняю, в какой из этих интервалов попало случайное число. Номер интервала — это и есть номер исхода. Могу показать по шагам. Пусть случайное число 0.37, тогда

```
  +\PH
0.3 0.5 1
  (+\PH)<0.37
1 0 0
  +/(+\PH)<0.37
  1++/(+\PH)<0.37
2
```

Число попало во второй интервал, значит реализовался исход «ОТКЛОНЕНО». Хуже другое, экспертная система по-прежнему дает слишком низкий процент правильных ответов.

Автор. Попробуем разобраться в чем дело. Программа и эксперт отличаются не только результатами, но и своим поведением в процессе их получения. Вашу программу совершенно не интересует то, что происходит в зале заседаний, кем поставлен и кем поддерживается вопрос, кто высказывается против, в какое время происходит обсуждение. Она с завидной невозмутимостью, которой так не хватает некоторым депутатам, «вытягивает из шапки» исходы с раз и навсегда заданными априорными вероятностями выбора. Вы посмотрите на нашего Эксперта.

В это время на рассмотрение Верховного Совета вынесли очередное постановление о деятельности кооперативов. Дядя Коля зачем-то заглянул в свои записи, пробежал глазами номер «Известий» и, как только председательствующий с экрана сообщил, что в прения записалось более 40 человек, уверенно сообщил: «Перенесут».

Читатель. Увы! Для принятия решения Эксперт, во-первых, использует более полные и разнообразные сведения об анализируемой системе и, во-вторых, применяет более регулярную процедуру получения заключений, в отличие от нашей горе-экспертной системы.

Автор (читателю шепотом). А вы попробуйте разговорить Эксперта и выяснить у него, какими именно знаниями он пользуется, предсказывая результаты рассмотрения вопроса. Кстати, при разработке экспертных систем эта деятельность, именуемая отторжением знаний эксперта, является важнейшей и наиболее трудоемкой. Ею занимаются специалисты, которых называют «инженеры по знаниям».

Читатель (Эксперту). Уважаемый дядя Коля, не могли бы вы пояснить, как вам удалось прийти к выводу о том, что вопрос будет перенесен?

Эксперт. Прежде всего я просмотрел свои записи и убедился, что вопрос обсуждается впервые. Я долго наблюдал за работой нашего парламента и обратил внимание на то, что очень часто решение впервые вынесенного на обсуждение вопроса откладывается на более поздний срок.

Читатель. И как же вы использовали эту информацию?

Эксперт. Если до получения ответа на этот вопрос моя уверенность в справедливости исходов составляла условно 30, 20 и 50 единиц, то теперь уверенность в третьем исходе (перенесение вопроса) в некоторой степени повышается, одновременно моя уверенность в том, что будут иметь место первые два исхода, несколько понижается.

Читатель. А дальше?

Эксперт. Затем, когда камера прошла по задним рядам палаты, я увидел, что на заседании присутствует необычно много людей, которые не являются членами Верховного Совета. Мне стало понятно, что вопрос имеет очень важное значение. И мое мнение относительно того, что вопрос будет отложен, еще больше укрепилось. К тому же, действие происходит на вечернем заседании, когда все устали и не склонны принимать серьезных решений.

Читатель. А в какой момент вы приняли окончательное решение?

Эксперт. Когда председательствующий сообщил, что в прения записалось более 40 человек, дальнейшие вопросы стали бессмысленными. Судите сами, очень важный вопрос, вечернее заседание и 40 человек в прениях. Да они просто не успеют все выступить!

Читатель. А могли ли вы ошибиться?

Эксперт. Несомненно. Проблема слишком сложна, чтобы гарантировать абсолютную точность. Даже самые знающие и опытные эксперты иногда ошибаются. Задача состоит в том, чтобы ошибаться, по возможности, реже и при этом не выходить за рамки ограничений со стороны заказчика экспертизы.

Читатель. Что это еще за ограничения?

Эксперт. Применительно к нашей задаче это прежде всего время упреждения прогноза. Действительно, чем более обширную информацию я соберу, тем более точен будет мой прогноз. Максимальный объем информации я буду иметь в момент постановки вопроса на голосование. Но скажите, пожалуйста, кому будет нужен мой прогноз, если результаты голосования через минуту будут известны всем? С другой стороны, если мой прогноз сделан заранее, то заказчик, например депутат, будет иметь некоторое время для принятия своего решения о необходимости выступить «за» или «против» данного законопроекта.

Читатель. Хорошо, по-человечески все более или менее понятно. Мне даже кажется, что если

бы я понаблюдал несколько месяцев за работой Совета, постоянно общаясь с вами и другими Экспертами из нашего двора, то я и сам, вероятно, научился бы предсказывать результаты голосования.

Эксперт (довольно сухо). Не сомневаюсь, что интерес к задаче, упорство и время позволят вам добиться успеха...

Читатель. Да, но наша цель совсем иная! Мы вот тут собрались, простите, запрограммировать вас, т. е. научить машину вашим методам работы. А как ее научишь тому, что после получения ответа на некоторый вопрос ваша уверенность «несколько (?!) возрастает»? А как запрограммировать, после каких ответов и на какие вопросы вы окончательно останавливаетесь на том или ином варианте исхода?

Эксперт. Простите, но я ничего не понимаю в компьютерах. Кроме того, я твердо убежден, что меня невозможно, как вы выразились, «запрограммировать». Так что все эти вопросы не ко мне. Разве что...

Читатель. Договаривайте, пожалуйста!

Эксперт. Не знаю, поможет ли это вам, но я всегда запоминаю, как часто положительный ответ на каждый вопрос, который я задаю сам себе, встречался при различных исходах голосования.

Читатель (воспрянув духом). Кажется, я понимаю! Например, во всех случаях, когда обсуждение законопроекта переносилось, для выступлений в прениях записалось больше 20 депутатов, и такое число никогда не встречалось при принятии или отклонении законопроекта. Теперь, когда вам стало известно, что в прениях собираются выступить более 40 депутатов, вы, конечно же, сказали, что обсуждение данного вопроса будет перенесено.

Эксперт. В общем, вы правильно поняли меня, но слишком упрощаете ситуацию. В такого рода задачах не бывает «во всех случаях» или «никогда». Бывает только «более или менее часто».

Читатель (упавшим голосом). Как же это измерить и учесть?

Эксперт. Лично у меня все это учитывается на уровне подсознания, которого, надо полагать, нет у вашей железяки. Но в знак моего глубочайшего сочувствия, я передаю вам на прощание список всех вопросов, которыми я задаюсь при проведении экспертиз. До свидания.

Читатель (Автору). Похоже, ваш хваленый АПЛ все же исчерпал свои возможности. Не будете же вы утверждать, что в нем имеется примитив, позволяющий учесть подсознание эксперта?

Автор. Конечно, нет. Таких средств нет ни в каком другом современном языке программирования. Но вы забыли, что мы создаем не естественный, а искусственный интеллект. Обратитесь к Крису Нейлору, он подскажет, что делать дальше.

Читатель. А где мне его найти?

Автор. На странице 62 в книге «Экспертные системы. Принципы работы и примеры», изданной издательством «Радио и связь» в 1987 году.

Читатель. Здравствуйте, господин Нейлор! Проконсультируйте меня, пожалуйста, по вопросам построения экспертной системы предсказания результатов голосования в парламенте.

К. Нейлор. О, это очень интересно! Слушаю ваш вопрос.

Читатель. Как представить в компьютере знания нашего эксперта о связи между ответами на интересующие его вопросы и исходом голосования?

К. Нейлор. Для вашего случая, пожалуй, подойдет представление знаний в виде условных вероятностей:

pp=p (ДА | И) и pм=p (ДА | не И)

Читатель. Что это за вероятности и откуда их взять?

К. Нейлор. Очень просто. Подсчитайте все случаи, когда на некоторый вопрос эксперта был получен ответ «ДА». Затем подсчитайте все случаи, когда реализовался некоторый исход И. Разделив первое число на второе, вы получите условную вероятность ответа «ДА» на данный вопрос эксперта при условии, что реализовался исход И. Например, законопроекты отклонялись 20 раз и в 5 случаях из этих 20 они рассматривались на вечернем заседании. Таким образом, условная вероятность ответа «ДА» на вопрос «Рассматривался ли законопроект на вечернем заседании парламента?» при условии, что он будет отклонен, равняется pp («Вечернее заседание?» — «ДА» | «ОТКЛОНЕН») = 5/20 = 0.25

Читатель. А вероятность pм?

К. Нейлор. Подсчитайте, сколько было случаев, когда законопроект не был отклонен (т.е. был принят или перенесен), и сколько раз при этом его рассмотрение происходило на вечернем заседании. Разделив второе на первое, получите значение pм.

Читатель. Ну ладно, а что теперь делать с этими вероятностями?

К. Нейлор. Теперь, получив ответ на некоторый вопрос, вы можете модифицировать первоначальные вероятности различных исходов. Это делается с использованием знаменитой теоремы Байеса, которая позволяет вычислить условную вероятность исхода И при том условии, что на некоторый вопрос эксперта был получен ответ «ДА»:

$$p\ (И\ |\ «ДА») = \frac{p(И)pp}{p(И)pp+(1-p(И))pм},$$

где p(И) — априорная (начальная) вероятность исхода И.

Читатель. Отлично! Тем самым мы моделируем то, что эксперт становится более уверенным в одних исходах и менее в других в зависимости от ответа на конкретный вопрос. Спасибо, господин Нейлор!

К. Нейлор. Пожалуйста, желаю успехов.

Читатель (сам себе). Ну, эту формулу я моментально запишу на АПЛ.

```
    R←PP БАЙЕС PM
[1] R←R÷(R←PN×PP)+PM×1-RN,
```

где PN — безусловная вероятность исхода...

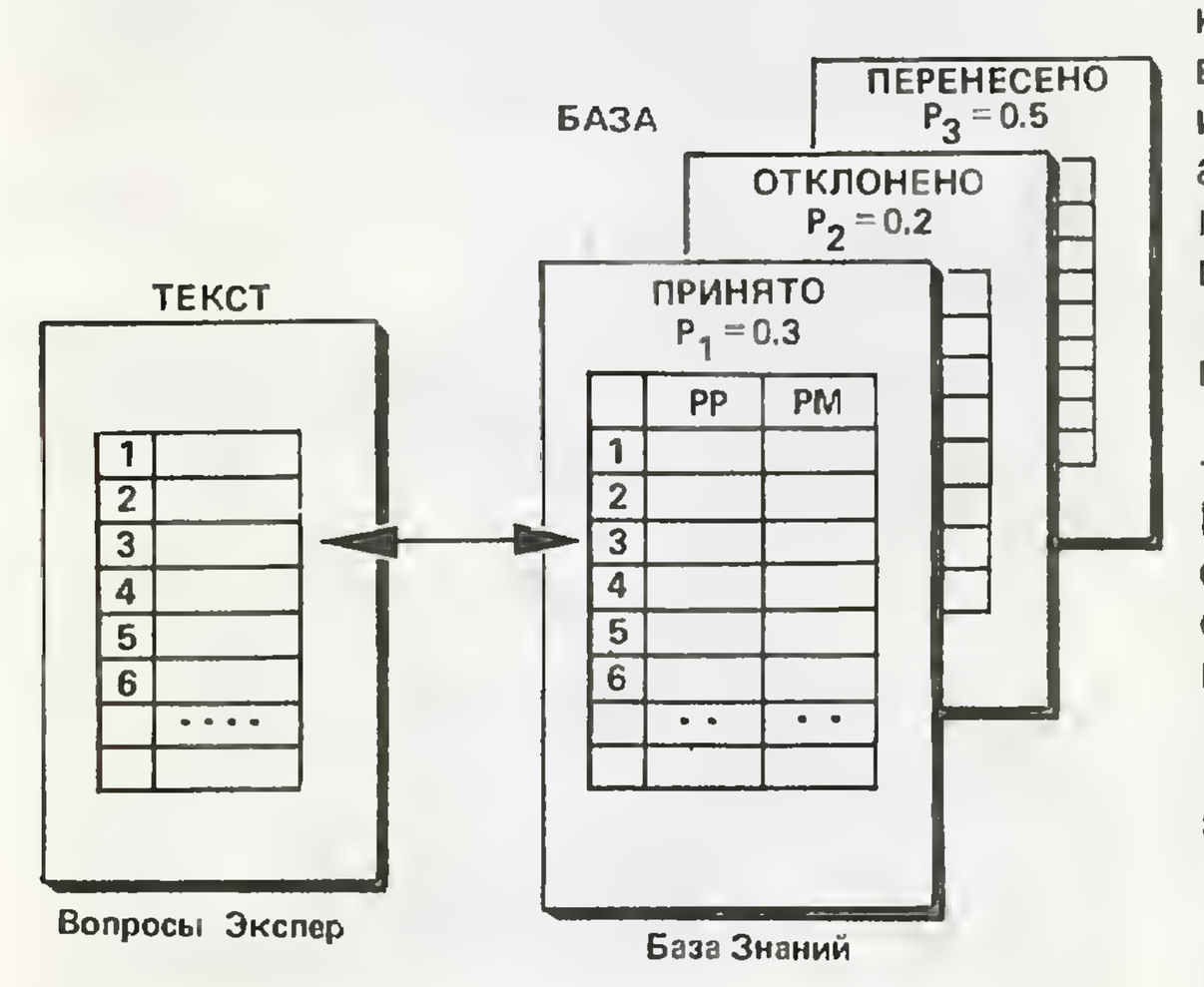
Автор. Не спешите! Формулу Байеса вы могли бы записать и на БЕЙСИКе. Помните, что наша задача много сложнее.

Читатель. Я следую советам господина Нейлора...

Автор. Правильно, но господин Нейлор объяснил вам, как модифицируется вероятность одного-единственного исхода при получении ответа на один-единственный вопрос! Вы же имеете три исхода и целый список вопросов, который дал вам наш Эксперт. Для любой пары «вопрос-исход» можно вычислить свои pp и pм. Все это и составит базу знаний вашей экспертной системы. Поэтому, прежде чем кодировать формулы, надобно задуматься о том, как всю эту информацию хранить, пополнять, изменять и использовать. Кстати, на некоторый вопрос эксперта может быть получен и ответ «НЕТ». В этом случае формула Байеса будет выглядеть так:

$$p\ (И\ |\ «НЕТ») = \frac{p(И)\ (1\text{-}pp)}{p(И)\ (1\text{-}pp) + (1\text{-}P(И))\ \ (1\text{-}pм)}$$

Читатель (после долгих размышлений). Придумал! Сначала я пронумерую все исходы. Собственно, это у нас уже сделано: номера строк матрицы ИСХОД, в которой мы храним результаты голосования, — это и есть их номера. Затем я пронумерую все вопросы из списка дяди Коли и организую символьную матрицу ТЕКСТ, в которой я буду хранить вопросы Эксперта: каждая строка — это вопрос из списка. Затем я возьму три листа бумаги и на одном из них сверху напишу ПРИНЯТО, на втором ОТКЛОНЕНО и на третьем — ПЕРЕНЕСЕНО. На каждом листе я нарисую вот такую таблицу.



Как вы, вероятно, догадались, эти три таблицы хранят pp и pм для всех пар «вопрос эксперта-исход голосования». Все эти таблицы можно представить в виде одного трехмерного массива АПЛ: первое измерение этого массива (назовем его БАЗА) отвечает за исходы голосования, второе — за вопросы Эксперта, а третье — за условные вероятности pp и pм.

Используя уникальные свойства АПЛ, я могу при получении ответа на некоторый вопрос подправить вероятности сразу всех трех гипотез. Например, если на пятый вопрос Эксперта был получен ответ «ДА», то новое распределение вероятностей исходов PN можно вычислить так:

```
PN←PH
PN←БАЗА[;5;1] БАЙЕС БАЗА[;5;2]
```

А при получении ответа «НЕТ» — вот так:

```
PN←(1−БАЗА[;5;1]БАЙЕС1—БАЗА[;5;2]
```

Далее я определю функцию ОТВЕТ, которая будет вычислять новое распределение вероятностей исходов при получении ответов «ДА» или «НЕТ» на вопрос Эксперта с номером J:

```
  ОТВЕТ J;PP;PM
[1]PP←БАЗА[;J[2];1]
[2]PM←БАЗА[;J[2];2]
[3]→(0>J[1]/M
[4]→0;PN←PP БАЙЕС PM
[5]M:PN←(1−PP) БАЙЕС 1−PM
```

Автор. Поясните.

Читатель. Аргументом этой функции является числовой вектор из двух элементов. Первый элемент — это ответ на вопрос: если он больше или равен 0, то это означает, что получен ответ «ДА», в противном случае мы считаем, что получен ответ «НЕТ». В зависимости от знака этого числа мы используем один из вариантов формулы Байеса, записанных в строке 4 (для ответа «ДА») или в строке 5 (для ответа «НЕТ»). Второй элемент аргумента — это номер вопроса. Используя свойства АПЛ, позволяющие работать с массивами так же, как и со склярами, мы подправляем вероятности сразу всех исходов, которые хранятся в векторе PN. Я сознательно ввел новое обозначение вектора вероятностей исходов для того, чтобы не испортить вектор PH априорных вероятностей, поскольку каждая экспертиза должна начинаться в ситуации, когда вероятности исходов есть значение PH.

Автор. Ну и как же вы теперь собираетесь проводить экспертизу?

Читатель. Очень просто! Я буду последовательно задавать все вопросы из списка дяди Коли, модифицируя значение вектора вероятностей гипотез. По исчерпанию списка я предскажу тот исход, вероятность которого будет максимальной.

Автор. Интересно будет посмотреть.

Читатель. Пожалуйста, вот программа моей экспертной системы:

```
  ЭКСПЕРТ; PN; B
[1] PN ← PH; B ← 1
[2] M:ОТВЕТ ЗАДАТЬ B
[3] → ((⍴ТЕКСТ) [1] ≥ B ← B + 1) / M
[4] 'Результат экспертизы:'
[5] (PN = Г / PN) / [1] ИСХОД
```

Сначала я запоминаю априорное распределение вероятностей исходов в векторе PN. Переменная B — это текущий номер вопроса из нашего списка. Строчки 2 и 3 программы будут выполняться столько раз, сколько строк имеет матрица вопросов ТЕКСТ. По исчерпанию списка вопросов мы определяем номер исхода, который имеет максимальную вероятность, и сообщаем его пользователю. Во второй строке я

использовал еще одну функцию — ЗАДАТЬ, которая задает вопрос с номером В и принимает ответ:

```
  R ← ЗАДАТЬ J
[1] R ← [], J ; [] ←ТЕКСТ[J;],'?'
```

Эта функция возвращает результат в формате, совпадающем с форматом аргумента функции ОТВЕТ. Кроме того, я определяю две переменные ДА и НЕТ, чтобы сделать работу с экспертной системой удобной:

```
  ДА ← 1
  НЕТ ← ¯1
```

Помните, что мы условились положительным числом кодировать ответ «ДА», отрицательным — «НЕТ». Теперь можно начать экспертизу:

```
  ЭКСПЕРТ
Обсуждение на вечернем заседании? ДА
В прения записалось более 20 человек? ДА
Председательствует М. С. Горбачев? НЕТ
. . . . . .
. . . . . .
. . . . . .
Результат экспертизы:
ПЕРЕНЕСЕНО
```

Читатель (с гордостью). Ну, как?

Автор (с определенной долей ехидства). Я бы предложил вам показать свое «произведение» нашему уважаемому Эксперту...

Сцена четвертая

На следующий день. Кажется, что в комнате никого нет, кроме компьютера. На самом деле там, как всегда, находится Автор. Входят Читатель и Эксперт. Все подходят к компьютеру. Читатель запускает экспертную систему. Эксперт, нацепив очки, внимательно изучает изображение на экране дисплея.

Эксперт. Поведение вашей системы все еще достаточно глупо. Я бы задал в три раза меньше вопросов, чтобы прийти к такому выводу. Поскольку вот уже на этом шаге (Эксперт ткнул пальцем в экран) мне все понятно, а ваша система продолжает задавать никому не нужные вопросы.

Читатель (Автору). Что же делать?

Автор. Спросите у г-на Нейлора.

Читатель. Г-н Нейлор, вы здесь?

К. Нейлор. Да, и с интересом слежу за развитием событий.

Читатель. И что же вы можете сказать на замечание Эксперта?

К. Нейлор. Эксперт совершенно прав. Но вы легко можете учесть его справедливое замечание. Предположим, система задала J вопросов, после которых вы имеете некоторое распределение вероятностей исходов PN. Давайте возьмем один из возможных исходов и рассчитаем вероятность PMAX этого исхода в предположении, что ответы на все пока еще не заданные вопросы будут свидетельствовать в его пользу. Аналогично можно вычислить вероятность PMIN этого исхода, предполагая, что все оставшиеся вопросы будет свидетельствовать против него.

Читатель. Это что же снова можно сделать с помощью теоремы Байеса?

К. Нейлор. Да. Не вдаваясь в подробности, скажу, что для независимых вопросов вероятность того, что на них одновременно будет получен ответ «ДА» при условии, что реализуется исход И, равно произведению соответствующих вероятностей pp:

p(ДА1 и ДА2 и ... ДАn | И) = pp = pp1·pp2·...·ppn

Это же справедливо и для вероятностей pm. Далее можно опять воспользоваться формулой Байеса, для того чтобы вычислить интересующие нас вероятность исхода И при условии, что на некоторые вопросы будет дан ответ «ДА». Однако хочу вас предостеречь. Дело в том, что нам нужны вероятности PMAX и PMIN, а в пользу (или против) исхода могут свидетельствовать как ответы «ДА», так и ответы «НЕТ»...

Читатель. Спасибо за подсказку. Я уже понял, что при вычислении вероятности PMAX необходимо в качестве pp использовать наибольшее из pp и 1-pp, а в качестве pm — минимальное из pm и 1-pm. Для PMIN я буду поступать наоборот. Но мне не ясно другое: что мы будем делать с этими вероятностями?

К. Нейлор. Представьте себе, что в некоторый момент вероятность PMIN одного из исходов стала такой, что она больше любой вероятности PMAX любого из остальных исходов. Это означает, что при любых ответах на оставшиеся вопросы, его окончательная вероятность PN все равно останется самой большой. Следовательно, задавать вопросы больше не имеет смысла. Именно так поступает и наш уважаемый Эксперт.

Эксперт (ничего не понял, но счел нужным сказать). Естественно...

Читатель. Прекрасно. Теперь я добавлю в нашу систему новый блок, который будет проверять необходимость продолжения процесса задавания вопросов. Для этого я определю такую функцию:

```
  R ← ПРОВЕРКА N; PMAX; PMIN; PP; PM
[1] PP←БАЗА [;N;1]
[2] PM←БАЗА [;N;2]
[3] PMAX←(x/PPГ1—PP)БАЙЕС x/PM L1−PM
[4] PMIN←(x/PP L1−PP)БАЙЕС x/PM Г1−PM
[5] →6×R←∧/R>(PMIN≠R←Г/PMIN)/PMAX
[6] PN←PMIN
```

Аргументом этой функции является вектор N, который хранит номера всех вопросов, которые еще не были заданы системой. Результат R — это 1, когда некоторый исход доминирует над другими, и 0 — в противоположном случае. В первом случае мы присваиваем значение вектора PMIN глобальному вектору PN текущего распределения вероятностей исходов. Это позволит системе обнаружить доминирующий исход и выдать его как результат экспертизы. Теперь наша система примет такой вид:

```
    ЭКСПЕРТ; PN; B
[1] PN←PH; B←⍳(⍴ ТЕКСТ)[1]
[2] M:→КОНЕЦ×⍳ПРОВЕРКА  B
[3] ОТВЕТ  ЗАДАТЬ  1↑B
[4] →(∅>⍴B←1↓B)/M
[5] КОНЕЦ: 'Результат экспертизы:'
[6] (PN=Г/PN)/[1]ИСХОД
```

Автор. Прокомментируйте, пожалуйста...

Читатель. Если в предыдущем варианте системы значение переменной B представляло собой номер текущего вопроса, то теперь это вектор номеров вопросов, которые еще не задавались системой. Очевидно, что в самом начале работы (строка 1) мы должны определить эту переменную как вектор целых чисел от 1 до N, где N — общее количество вопросов. Каждый цикл работы системы начинается с поиска доминирующего исхода. Если такой находится, то работа прекращается (переход со строки 2 на строку 5). В противном случае мы задаем очередной вопрос и модифицируем вероятности исходов. В 4-й строке мы выбрасываем из вектора B номер заданного вопроса и, если этот вектор не пуст, повторяем процедуру. Если заданы все вопросы (в векторе B больше нет элементов), то работа прекращается и выдается название исхода, имеющего максимальную вероятность. Надеюсь, теперь вы, дядя Коля, признаете интеллектуальные способности системы?

Эксперт. Фи! Эта ваша система может посоревноваться, разве что, с моим трехлетним внуком. Ведь она задает вопросы совершенно бессистемно. Вместо того чтобы сразу начинать с главного, она интересуется сначала второстепенными вопросами. Получив ответ, свидетельствующий в пользу некоторого исхода, она может перескочить на другую тему, не имеющую к нему никакого отношения. Я же действую целенаправленно и последовательно.

Читатель. Г-н Нейлор, вы можете проконсультировать меня еще раз?

К. Нейлор. С удовольствием! Важность вопроса для одного данного исхода можно оценить абсолютной величиной разности вероятностей этого исхода после получения ответов «ДА» и «НЕТ». Важность вопроса может изменяться от 0 для неинформативных вопросов до 1 для вопросов, которые однозначно подтверждают или отвергают данный исход.

Читатель. Все понятно. Сейчас я напишу на АПЛ еще одну функцию, которая будет выбирать наилучший вопрос среди всех, что еще не задавались.

```
    R←ВОПРОС; PP; PM; РДА; PHET; ВЕС
[1] PP←БАЗА [;B;1]
[2] PM←БАЗА [;B; 2]
[3] РДА←PP  БАЙЕС  PM
[4] PHET←(1—PP) БАЙЕС  1—PM
[5] R←1↑B ВЕС ⍳Г/ВЕС←+/[1]|РДА—PHET]
[6] B←(P≠B)/B
```

Функция ВОПРОС устроена так. Сначала с помощью формулы Байеса мы вычисляем условные вероятности пар «вопрос-исход» для всех вопросов, которые еще не были заданы, в предположении, что на эти вопросы будет получен ответ «ДА» (матрица РДА) и «НЕТ» (матрица PHET). Номера всех еще не заданных вопросов хранятся в глобальном векторе B. Затем мы вычисляем вектор весов вопросов ВЕС, каждый элемент которого характеризует важность соответствующего вопроса. В качестве результата функции ВОПРОС выдаем номер того из них, который имеет максимальный вес по всем исходам. После этого выбрасываем номер этого вопроса из вектора B. Теперь наша система приобретет следующий вид:

```
    ЭКСПЕРТ; PN; B
[1] PN←PH; B←r(⍴ТЕКСТ) [1]
[2] M:→КОНЕЦ×⍳ПРОВЕРКА  B
[3] ОТВЕТ  ЗАДАТЬ  ВОПРОС
[4] →(∅>⍴B)/M
[5] КОНЕЦ: 'Результат экспертизы:'
[6] (PN=Г/PN)/[1] ИСХОД
```

По сравнению с предыдущим этот вариант отличается тем, что на каждом шаге мы будем задавать уже самые значимые вопросы!

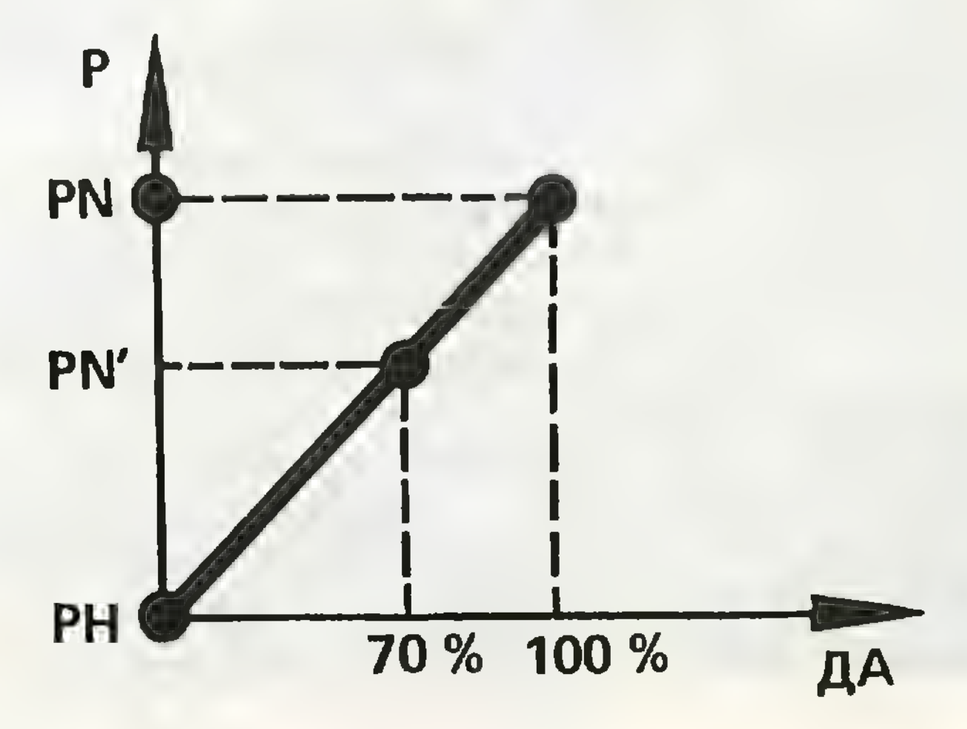
К. Нейлор. Блестяще!

Автор. Неплохо.

Эксперт. Подождите радоваться. Не на каждый вопрос можно дать четкий ответ. Вот, к примеру, я обратил внимание на то, что члены Верховного Совета обычно прислушиваются к мнению депутата С. Поэтому в моем списке вопросов есть вопрос: «Поддерживает ли депутат С. этот законопроект?» Ну а если мне неизвестно его отношение к данному вопросу? Что же я должен ответить вашей системе? Или вот вчера был случай. Депутат С. выступал в прениях по одному из постановлений, но, прослушав все до конца, я так и не понял точно — поддерживает он постановление или нет. Вчера я оценил его выступление как «ДА» на 30%. Интересно, как вы сможете учесть такие ответы?

К. Нейлор. Линейная интерполяция...

Читатель. Точно! Уважаемый Эксперт, сейчас мы учтем и это ваше замечание. Посмотрите на этот рисунок. Здесь я предположил, что при ответе «ДА» вероятность PN некоторого исхода увеличивается, впрочем, не это важно. Когда мы получаем ответ «твердое ДА», мы изменяем исходное PN на некоторую величину, используя формулу Байеса. Но если мы получаем ответ «ДА на 70%», то мы можем учесть нечеткость ответа с помощью метода линейной интерполя-



ции так, как это показано на рисунке. То есть:

PN←PN+0.7×(PP БАЙЕС PM)—PN

Как вы, очевидно, помните, переменные ДА и НЕТ — это числа 1 и ¯1 соответственно. Во-первых, мы определим новую переменную:

НЕИЗВЕСТНО ← 0

А во-вторых, разрешим вводить ответы вида 0.7×ДА или 0.2×НЕТ, что означает «ДА на 70%» и «НЕТ на 20%». Кстати, когда ответ на какой-либо вопрос неизвестен, то можно ввести ДА+НЕТ, что тоже достаточно мнемонично. Теперь мы слегка подправим функцию ОТВЕТ, чтобы в ней могли учитываться нечеткие ответы:

```
      ОТВЕТ J; PP; PM
[1]  PP←БАЗА [;J;1]
[2]  PM←БАЗА [;J;2]
[3]  →(∅>J[1]/M
[4]  →∅;PN←PN+J[2]×(PP БАЙЕС PM)—PN
[5]  M:PN←PN+J[2]×((1−PP)БАЙЕС 1−PM)−PN
```

Эксперт. Мне больше нечего возразить.

Автор. Теперь надо приступить к эксплуатации системы, поскольку только практика может дать ей окончательную оценку.

Сцена пятая

Читатель исчезает на несколько месяцев. До Автора доходят слухи, что система дает обоснованные предсказания и ошибается не чаще Эксперта. Автор и Читатель снова встречаются в той же комнате.

Автор. Странно. Я ожидал увидеть сияющего победителя, а на вашем лице опять озабоченность.

Читатель. Первые восторги давно прошли. Система действительно показала себя неплохо. Однако с течением времени она как будто «износилась». Стали появляться грубые ошибки. А наш уважаемый Эксперт, наоборот, так «насобачился» в этом деле, что выдает все более и более точные предсказания.

Автор: Этого следовало ожидать, ведь Верховный Совет не является застывшей системой. Меняется обстановка, меняются люди, а ваша программа по-прежнему пользуется полученными когда-то неизменными знаниями. С другой стороны, Эксперт постоянно учится, приобретает новый опыт, расширяет и углубляет свои знания. Блок обучения является важнейшей частью любой экспертной системы, а у вас он отсутствует полностью.

Читатель. Похоже, вы правы. Пойду схожу за дядей Колей. Надо выяснить, как он получает новые знания.

Через некоторое время Читатель возвращается с Экспертом, на ходу излагая суть возникших трудностей.

Эксперт. Мое обучение складывается из нескольких компонент. Ну, например, я обращаю внимание на то, что вырос авторитет депутата X, мнением которого я раньше не интересовался. Я замечаю, что у него особенно развит критический дар, и если он выступает против какой-либо резолюции, то довольно часто эта резолюция отклоняется.

Читатель. С приобретением таких знаний моя система справится просто. Необходимо собрать данные о том, как часто депутат X выступал против какого-либо решения и оно при этом отклонялось. Также необходимо выяснить, сколько раз Верховный Совет принимал другие решения относительно обсуждаемого вопроса, несмотря на то что депутат X выступал против. Все эти данные позволят вычислить условные вероятности pp и pm, относящиеся к этому новому вопросу, который будет задавать система. Базу знаний можно легко дополнить так:

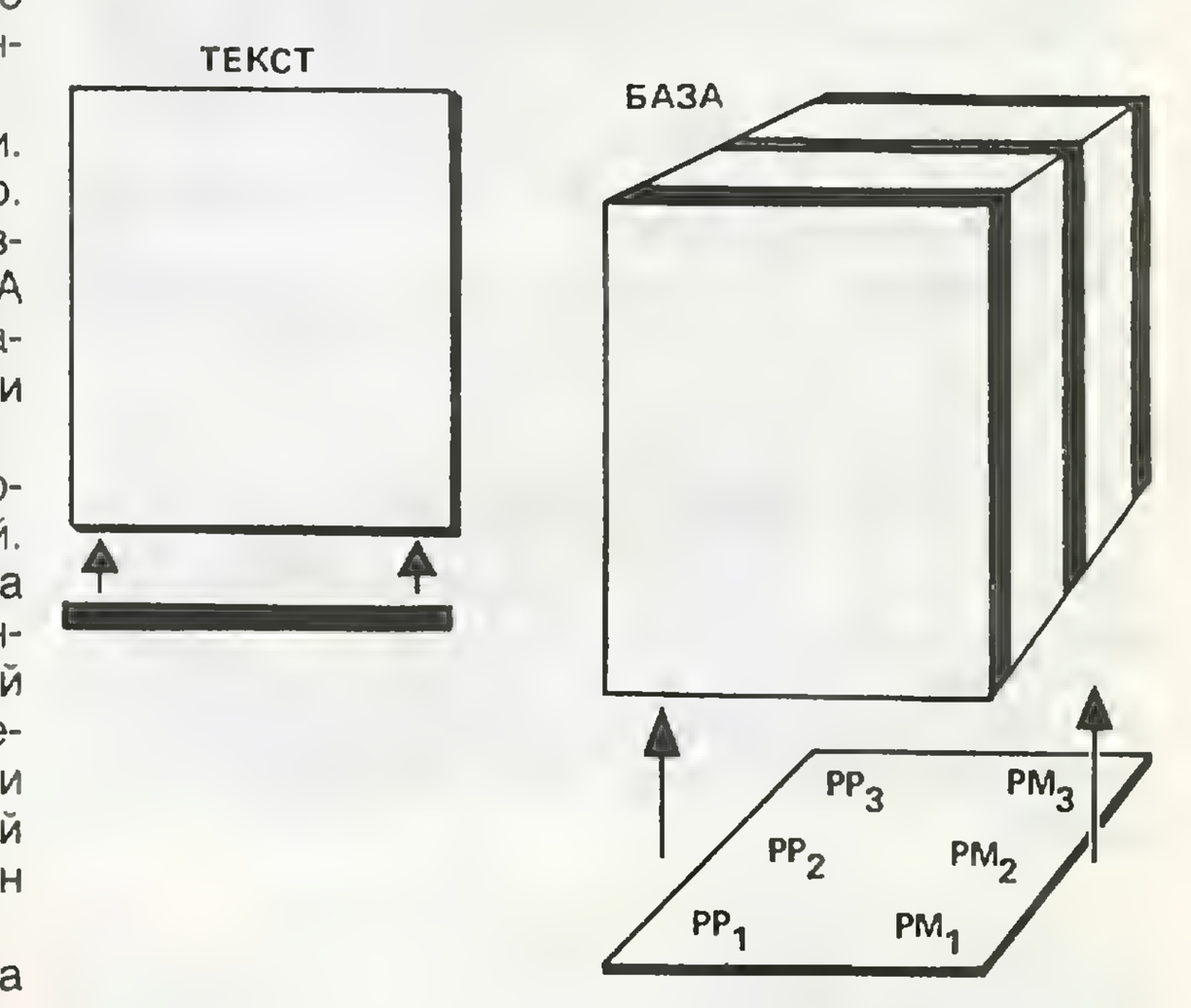
БАЗА←БАЗА, [2]3 2⍴PP1, PM1, PP2, PM2, PP3, PM3

А матрицу вопросов так:

ТЕКСТ←ТЕКСТ, [1] 1 ↓ ⍴ТЕКСТ) ↑ 'Выступает ли депутат X против'

Эксперт. Что-то я ничего здесь не пойму.

Читатель. Первое выражение «подклеивает» к нашему трехмерному массиву БАЗА снизу еще один слой, относящийся к новому вопросу. Значок «,» (запятая) обозначает очень мощный двухместный примитив «конкатенация» (объединение). Таким образом, я сначала сформировал матрицу из условных вероятностей исходов, а затем приклеил ее к массиву БАЗА, указав в квадратных скобках, что это необходимо сделать по второму измерению. Примерно так же я поступил и с матрицей вопросов. Однако во втором выражении я использовал примитив «взять» для того, чтобы подклеить к матрице новую строку подходящей длины.



Эксперт. Вторым моментом в моем обучении является то, что я слежу за изменением частот принятия различных решений. Я заметил, что по мере накопления опыта Верховный Совет стал более решительным и вопросы стали реже переноситься. Выросло доверие к мнению комитетов, которые стали выносить на пленарные слушания в основном хорошо проработанные законопроекты.

Читатель. Что ж, и этот фактор я могу легко учесть. Речь идет о том, что со временем изменяется вектор априорных вероятностей РН. Я заведу специальный числовой вектор ИТОГИ, состоящий из трех элементов, в котором буду учитывать фактические итоги решений тех или иных вопросов. Элемент ИТОГИ [1] будет хранить количество решений «ПРИНЯТО», а элементы ИТОГИ [2] и ИТОГИ [3] — «ОТКЛОНЕНО» и «ПЕРЕНЕСЕНО». Если при рассмотрении некоторого вопроса Верховный Совет принимает решение «ОТКЛОНЕНО», то я прибавляю 1 ко второму элементу этого вектора:

ИТОГИ [2] ← ИТОГИ [2] + 1

Новые априорные вероятности вычисляются так:

РН←ИТОГИ÷ ÷ /ИТОГИ

Эксперт. В-третьих, меняется мое отношение к ответам на вопросы, которые я задаю при проведении экспертизы. Например, несколько недель назад я заменил свой старый черно-белый телевизор на новый — цветной. Во время первой же трансляции заседания Верховного Совета я обратил внимание на то, что депутат Y пришел на него в зеленом галстуке. Я с детства не люблю зеленых галстуков, но плюрализм есть плюрализм. На том заседании были рассмотрены два постановления, и оба были приняты. На следующий день мне опять на глаза попался этот галстук, и снова голосование завершилось положительно. Естественно, что я с нетерпением ждал очередной трансляции. Представьте себе мое изумление, когда Y снял, наконец, свой дурацкий галстук, и в этот день решения по всем обсуждавшимся вопросам были отложены! К тому моменту, когда я пришел к выводу о влиянии цвета галстука депутата Y на результаты голосований, всего состоялось 10 различных решений, из которых 5 — «ПРИНЯТО», и во всех случаях Y был в зеленом галстуке. С другой стороны, когда принимались другие решения, Y был в галстуке другого цвета. Пользуясь вашей терминологией, рр исхода «ПРИНЯТО», при условии наличия зеленого галстука, равна 0.5, а рм — 0. Когда Y опять появился в зеленом галстуке, я моментально предсказал принятие законопроекта, и оказался прав. Конечно, потом пришлось разочароваться. Дело было не в галстуке, а в случайном совпадении. Понаблюдав достаточно долго, я убедился, что решения нашего парламента, слава богу, не зависят от цвета галстука депутата Y. Могу привести вам и массу других примеров, когда ценность моих знаний уменьшалась или, наоборот, увеличивалась.

Читатель. Спасибо. Мне стало понятно, что вероятности рр и рм также необходимо обновлять в процессе работы системы. Я даже придумал, как можно это сделать. Я добавлю к моей базе знаний еще один столбец, то есть теперь каждой паре «вопрос—исход» будут соответствовать не два числа — рр и рм, а три: рр, рм и число случаев, когда на вопрос давался ответ «ДА» и фактически имел место данный исход. Зная полную раскладку результатов голосований, которые я накапливаю в векторе ИТОГИ, и эти числа, я легко могу пересчитать вероятности рр и рм!

Автор. Отлично! Теперь можно сказать, что экспертная система процентов на 90 готова.

Читатель (с обидой). Почему только на 90?

Автор. Ну, во-первых, очень важной функцией любой экспертной системы является ее способность объяснять свои выводы. Этим во многом определяется доверие пользователя к системе. Во-вторых, в вашей системе всего три возможных исхода и не очень много вопросов. Поэтому представление базы знаний в виде регулярного массива здесь приемлемо. Но ведь обычно экспертные системы имеют дело с десятками, а то и сотнями исходов и с еще большим количеством вопросов. При этом есть вопросы, которые имеют отношение не ко всем, а только к некоторым исходам. Вам же придется хранить совершенно бесполезную информацию, расходуя память ЭВМ. В таких случаях для хранения знаний можно воспользоваться структурами, называемыми списками.

Читатель. Освоив такое мощное средство, как АПЛ, я уверен, что смогу справиться и с этими проблемами. Например...

ЗАНАВЕС

(Окончание)

мантиссы, включая точку. Для приведенного в статье формата %8.04 регистр R2 содержит (числа восьмеричные):

ст.байт = 10
мл.байт = (4 + 1)*2 = 12
R2 = <10><12> = 4012

3. Многие пользователи проявили заинтересованность в применении плавающей арифметики на ЭВМ с Бейсик-интерпретатором. К сожалению, авторы не занимались изучением заложенных в микросхеме БЕЙСИКа средств плавающей арифметики. Однако ими разработан пакет подпрограмм, который позволяет использовать механизм псевдокоманд плавающей арифметики ФОКАЛа на любом БК, в том числе и на ЭВМ с БЕЙСИКом. В состав пакета входит также библиотека стандартных математических функций.

В заключение авторы хотят выразить благодарность всем пользователям, уже высказавшим свои замечания, соображения и доработки. Все замечания и предложения просим направлять по адресу: 226063, г.Рига, ул.Кенгарага 8, НИИ Физики твердого тела.

Гвоздев С.
Эрнестсонс Г.

Л. Н. ЖАРИКОВ

ВЫБОР МАГНИТОФОНА ДЛЯ БК 0010

В качестве внешнего запоминающего устройства бытового компьютера БК 0010 может быть использован практически любой магнитофон, но для более надежной работы при записи и считывании файлов (программ и данных) на магнитофоне следует отключить автоматическую регулировку уровня записи (АРУ). АРУ может мешать из-за вытягивания уровня сигнала там, где это нежелательно.

Руководство по эксплуатации БК 0010 рекомендует при записи файлов на магнитную ленту устанавливать максимальный уровень усиления, но опыт использования БК с магнитофоном показывает наличие оптимального уровня, своего для каждого магнитофона. Его можно установить после нескольких попыток с помощью регулятора уровня записи. Обычно оптимальный уровень достигается при некотором превышении уровня записи по сравнению с записью обычных фонограмм. Выходное напряжение на линейном выходе магнитофона при этом составляет 0,5 — 0,8 В.

При подготовке к работе магнитофона с компьютером прежде всего убедитесь в его исправности и хорошей работе на обычных записях. Звук не должен плавать, не должно быть тресков и других лишних шумов. Магнитную ленту не должно заедать. Чтобы отбраковать кассету, поставьте ее в магнитофон и прокрутите в обе стороны. Если лента перемотается бесшумно, то ее можно использовать. Если будут слышны скрип или царапанье, то она будет вас подводить. Иногда избавиться на время от скрипа помогает смазывание спиртом поролоновой прокладки на кассете, но лучше скрипящую кассету при работе с компьютером не использовать.

Если старая запись плохо стерта, то это также может быть причиной сбоев при чтении файлов. Лучше всего предварительно стирать старую запись, особенно если она далеко не первая на этом месте.

Плохой лентопротяжный механизм и некачественный магнитный слой тоже могут стать причиной ненадежной работы магнитофона с БК 0010.

Часто причиной сбоя могут быть помехи в электрической сети, возникающие при включении и выключении мощных электрических приборов, даже обычного утюга. Кроме того, свою лепту могут вносить помехи от телевизора (особенно от цветного) и от теле- и радиостанций. Источником сбоя может стать даже проезжающий автомобиль с неисправным зажиганием.

Чтобы проверить, не является ли телевизор причиной сбоев, после полного набора команды чтения файла выключите телевизор и затем нажмите клавишу ввода. Когда вы убедитесь на слух, что файл на ленте окончился, остановите магнитофон и включите телевизор. Результат чтения файла вы увидите на экране.

Все перечисленные выше рекомендации касаются работы со своим магнитофоном и магнитной лентой, записанной на этом магнитофоне. Обычно при соблюдении указанных правил исправный магнитофон работает с БК 0010 достаточно надежно.

Работа с МЛ, записанными на других магнитофонах

Основные проблемы при работе с магнитофоном начинаются, когда вы пытаетесь прочитать файл, записанный на чужом магнитофоне. Часто возникающая несовместимость головок записи-чтения приводит к тому, что единственным способом прочтения файла может стать подстройка считывающей головки магнитофона, хотя выполнять ее нужно только в крайнем случае. Некоторые пользователи даже имеют для таких ситуаций отдельный магнитофон.

Подстройка считывающей головки обычно выполняется следующим образом. Запоминают положение регулирующего винта считывающей головки. При фиксированном положении ручки «Громкость» и нажатой клавише «Воспроизведение», вращая регулирующий винт, необходимо добиться максимальной громкости звука. После этого операцию считывания файла можно повторить. В случае стереофонического магнитофона вращение регулирующего винта в режиме воспроизведения может оказаться невозможным из-за другого устройства приемника кассеты. В этом варианте придется многократно вынимать и вставлять кассету.

Не забудьте после успешного считывания файла вернуть считывающую головку в исходное положение! Гораздо лучше, если все ваши файлы будут записаны при одном положении головки магнитофона. Файлы, записанные при других положениях головки, легко распознаются по глухому звучанию записи и невозможности их ввести в компьютер без смещения головки.

Если запись при всех положениях считывающей головки не удается прочитать, попробуйте

ввести ее с одного из каналов стереомагнитофона. Это наиболее надежный способ. Если и это не помогает, а файл является очень ценным, воспользуйтесь советом А. Константинова, приведенным в конце.

Организация хранения и поиска информации на магнитной ленте

Если на первых порах использования БК с магнитофоном при небольшом количестве кассет вы сможете хранить файлы бессистемно, то через некоторое время необходимость какой-то системы в хранении станет очевидной. Помнить, на какой кассете, на какой стороне, в каком месте ленты какая программа хранится, на каком языке, как запускается и т. п. без специального каталога трудно.

Желательно программы, написанные на различных языках, хранить на разных кассетах.

Одним из способов организации файлов на магнитной ленте является использование речевых меток файлов. Совмещая стандартную процедуру записи программ на МЛ и использование микрофона, можно перед каждым файлом помещать соответствующую ему речевую информацию. При поиске файла достаточно включить воспроизведение и установить громкость. Запись речевых меток на МЛ можно выполнять в стандартном режиме записи используемого магнитофона.

Часто на магнитофоне устанавливается счетчик метража МЛ. Наличие счетчика позволяет вести рукописный каталог, в который заносятся имена файлов, показания счетчика относительно начала МЛ и другая необходимая информация. Такой каталог позволяет быстро находить требуемый файл, используя ускоренную перемотку МЛ. Пример формы каталога представлен в таблице.

Таблица

Форма ведения рукописного каталога файлов

Кассета CAMES				Сторона 1 _____		
Место	Имя файла	Регистр	Язык	Адрес загрузки	Адрес запуска	Примечание
0	MAPC	Лат	Коды	1000	5554	Запуск из тестов
5	KING	Лат	ФОКАЛ	—	—	Президент
10	ТАРЗАН	Рус	Коды	760	Авт.	Автозапуск из монитора
16	КМ	Рус	ФОКАЛ	—	—	Компьютерный мир

Если на вашем магнитофоне нет счетчика метража, в любом случае заносите в каталог пометки о месте, ориентируясь на деления, имеющиеся на кассете. Это тоже существенно уменьшит время поиска файлов. Для рабочих кассет каталог ведите карандашом, чтобы можно было быстро обновлять пометки.

Как сэкономить магнитную ленту

Известно, что всякая экономия сводится к экономии времени. В нашем случае к экономии времени на зарабатывание денег для покупки кооперативных кассет или времени на поиск их на пустых полках радиомагазинов.

Помните, что любая экономия магнитной ленты хороша только на хорошо работающем магнитофоне!

Если вы хотите сэкономить пленку и время считывания файлов, то перепишите нужные файлы с помощью трехскоростной программы копирования КОПИР-М. В случае использования первой скорости копирования достигается двойное сжатие файлов. БК 0010 достаточно надежно считывает такие файлы.

Если вы не имеете этой программы, не отчаивайтесь! Перепишите файл на другой магнитофон на пониженной скорости, а затем запишите обратно на повышенной. Получится требуемое сжатие записи.

Ни в коем случае не экономьте на дублях записей! Любой нужный файл должен быть записан по крайней мере дважды, а ненужные стирайте сразу!

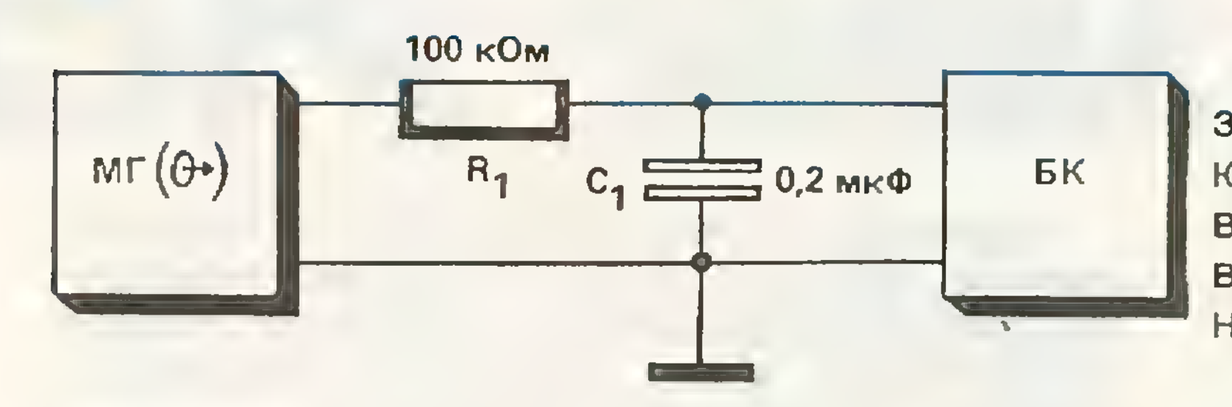
Сэкономить магнитную ленту при работе на БЕЙСИКе БК 0010. 01 можно, записывая программу командой CSAVE. В этом случае программа занимает существенно меньше места. Команда SAVE может пригодиться только для того, чтобы считывать затем программу в память в качестве данных с помощью оператора INPUT # и дальнейшего ее редактирования вашей программой.

Повышение надежности хранения информации

Основным способом повышения надежности хранения информации на магнитной ленте является дублирование файлов.

Храните кассеты и бобины подальше от источников электромагнитных полей.

Кроме всех тех приемов, которые описаны выше, можно использовать технические способы. В частности, параллельно шнуру для записи на МГ можно подключить керамический конден-



сатор емкостью 0,1—0,2 мкФ.

Иногда хорошего качества считывания файлов добиваются, снимая сигнал не с линейного выхода магнитофона, а с выхода, предназначенного для наушников или дополнительного динамика.

При возникновении сбоев при считывании ранее хорошо читающихся файлов протрите считывающую головку магнитофона спиртом или шрифтоочистителем и при успешном чтении тут же продублируйте файл. Это правило позволит вам сэкономить много нервов.

Особые случаи работы с магнитофоном

Довольно часто в работе на бытовом компьютере приходится сталкиваться с чтением текстовой информации с магнитофона. Такой информацией может быть, например, файл документации по какой-либо программе, где несколько ошибок в символах могут несущественно влиять на потребительскую ценность текста. Для чтения подобных текстовых файлов можно разработать не очень сложную программу в кодах, блокирующую возникновение ошибки контрольной суммы файла. Эта программа была бы очень полезна и для восстановления плохо читающихся программ на алгоритмических языках. Разработкой подобной программы могут заняться любители программирования на АССЕМБЛЕРе и в машинных кодах.

Другим особым случаем работы с магнитофоном является ввод в БК 0010 речевой информации. Пользователи, имеющие программу РЕЧЬ-2, наверняка встречались с тем, что в промежутках между словами при воспроизведении речи, записанной в компьютер, возникают сильные шумы. Это связано с генератором подмагничивания магнитофона. Высокочастотный сигнал подмагничивания головки имеет большую амплитуду на частоте около 75кГц и, несмотря на фильтры магнитофона, пробивается на линейный выход при записи информации с микрофона.

В обычном режиме записи магнитофона эта высокочастотная составляющая в динамике не воспроизводится, так как находится за пределами возможностей как динамиков, так и человеческого уха. В случае воспроизведения речи программой компьютера, использующей при кодировании клиппирование сигнала (ограничение его амплитуды), сигнал высокой частоты представлен в зашифрованной информации относительно большим количеством единичек в памяти по сравнению с низкочастотным, т. е. бесполезный для нас сигнал забивает полезную информацию уже при кодировании речи в компьютере.

Уменьшить эту помеху от магнитофона при записи поможет дополнительный фильтр, срезающий высокие частоты в полосе около 75 кГц на выходе магнитофона. Для этого нужно вставить в кабель магнитофона RC-цепочку, показанную на рисунке.

Полезные советы:

— если добиться надежной работы с магнитофоном не удается, посоветуйтесь с более опытным пользователем или сходите в ближайший клуб пользователей персональных компьютеров;

— проверить, хорошо ли БК 0010 стыкуется с магнитофоном, можно с помощью способа, предложенного А. Погаевским (пос. Провидение) в журнале «Информатика и образование», 1989, № 3;

— выбирая магнитофон для работы с БК 0010, И. Гарячук (г. Балаково) советует придерживаться принципа «чем ниже класс, тем лучше». Из схемы АРУ магнитофона «Протон-402» он удалил транзистор, чтобы разорвать цепь АРУ (Информатика и образование, 1989, № 3);

— если у вас нет кассетного магнитофона, пригоден и бобинный;

— при использовании бобинного магнитофона может пригодиться схема соединительного кабеля, предложенного М. Максимовым (г. Москва), (Наука и жизнь, 1986, № 6);

— если программа не считывается, ее все же можно «вытянуть» с кассеты, воспользовавшись двумя магнитофонами. С кассетного магнитофона нужно переписать программу на катушечный на скорости 19 см/с, а считывать в память компьютера на скорости 4 см/с. Восстановленную таким образом программу можно, как обычно, переслать из памяти БК на кассету (А. Константинов, г. Свердловск, журнал «Наука и жизнь», 1987, № 10);

— производите промежуточную запись файлов во время их ввода с пульта.

Можно установить за правило записывать, например, при добавлении каждых 40 строк. Это избавит вас от повторения набора файла при неожиданных отказах.

Литература

1. Осетинский Л. Г. и др. ФОКАЛ для микро- и мини-компьютеров. — Л., Машиностроение, 1988.

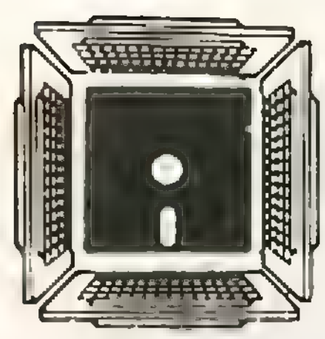
2. Кузьмин Ю. Я. Бытовой компьютер БК 0010 — надежды и возможности. Вычислительная техника и ее применение. — 1988. — № 7.

3. Наука и жизнь. — 1986. — № 6. — 1987. — № 10.

4. Информатика и образование — 1988. — № 6; 1989. — № 3.

5. «Электроника БК 0010-01». Руководство по эксплуатации.

БК ЗА РОГА



В. Д. БУРТЯНСКИЙ, студент ДГУ
г. Днепропетровск

Выпускаемые ныне компьютеры БК-0010-01 оснащаются транслятором БЕЙСИКа версии 1986.07-24, которая обладает существенным недостатком: в ней не реализована команда MERGE, позволяющая объединять программы, записанные на магнитной ленте. Вследствие этого затрудняются создание и использование библиотеки подпрограмм.

Этот недостаток можно преодолеть: достаточно воспользоваться приводимой ниже короткой программой в кодах, включаемой в Бейсик-систему. Программу можно разместить в любом свободном участке адресного пространства длиной & 0 144 байта, например в области стека с адреса & 0 400 (если есть уверенность, что не произойдет взаимного перекрытия программы и стека) или вблизи верхней границы ОЗУ, выделив для этого свободный участок памяти оператором CLEAR с соответствующими параметрами.

После загрузки необходимо запустить программу с начального адреса. Можно объединить загрузку и запуск, пользуясь командой

BLOAD «имя», R.

После запуска появляется возможность использовать новую команду:

MERGE «спецификация файла»,

которая загружает с ленты программу на БЕЙСИКе, записанную в формате ASC, и объединяет ее с программой, хранящейся в ОЗУ. Если номер какой-либо строки загруженной программы совпадает с номером одной из строк, хранящихся в памяти, то загруженная строка замещает старую.

Исключить рассматриваемую программу из Бейсик-системы можно командой:

POKE&0 30, &0 100112

Принцип работы программы заключается в следующем. Будем считать, что начальный адрес равен 1000. После запуска вектор прерываний по команде EMT устанавливается на адрес 1014 и производится останов. По адресам 1014—1050 расположен диспетчер команды EMT, который обеспечивает обработку EMT-прерываний и отлавливает прерывания по команде EMT 6, при которых с клавиатуры был введен символ <BK> (код &0 12). Если такая ситуация возникла, то анализируется содержимое буфера ввода Бейсик-системы, расположенного с адреса 2422 (адреса 1060—1120). Если там не находится команда MERGE, то осуществляется возврат из прерывания. В противном случае символ <BK> печатается и добавляется к содержимому буфера, восстанавливается указатель стека, гасится курсор и управление передается в ПЗУ по адресу 136116 — это другой вход в программу обработки команды LOAD.

Текст программы в кодах и на АССЕМБЛЕРе:

```
1000  010737    MOV  R7, @ #30
1002     30
1004  062737    ADD #12, @ #30
1006     12
1010     30
1012  000000    HALT
1014  010546    MOV  R5, –(R6)
1016  016605    MOV  2(R6), R5
1020      2
1022  014505    MOV  - (R5), R5
1024  042705    BIC #177400, R5
1026  177400
1030  010546    MOV  R5, - (R6)
1032  016505    MOV 100000(R5), R5
1034  100000
1036  004715    JSR  R7, @R5
1040  022627    CMP  (R6)+, # 6
1042      6
1044  001402    BEQ    A
1046  012605    B: MOV  (R6)+, R5
1050  000002       RTI
1052  120027    A:  CMPB  RO, #12
1054     12
1056  001373      BNE   B
1060  012705      MOV #2422, R5
1062   2422
1064  122527     CMPB (R5)+, # 115
1066   115
1070  001366      BNE   B
1072  122527     CMPB (R5)+,  # 105
1074     105
1076  001363     BNE   B
1100  122527     CMPB  (R5)+, # 122
1102   122
1104  001360     BNE    B
1106  122527     CMPB (R5)+, # 107
1110   107
1112  001355     BNE   B
```

```
1114  122527   CMPB (R5)+, #105
1116    105
1120  001352   BNE  B
1122  104016   EMT 16
1124  110021   MOV B R0, (R1)+
1126  062706   ADD  #6, R6
1130     6
1132  010503   MOV   R5, R3
1134  004737   JSR   R7,@# 124120
1136  124120
1140  000137   JMP @# 136116
1142  136116
```

Комментарий рецензента на программу, реализующую возможности команды MERGE БЕЙСИКа БК0010.01

В версии БЕЙСИКа БК0010.01 отсутствует очень полезная команда MERGE, позволяющая в других версиях объединить программу, находящуюся в ОЗУ, с программой, загружаемой с внешнего устройства. Вследствие этого затрудняется накопление пользователями библиотек подпрограмм.

Предложенная автором программа позволяет выполнять команду MERGE "имя файла", объединяя программу в ОЗУ с программой на магнитной ленте. Команда должна вводиться в диалоговом режиме, обязательно без пробела между кавычкой и именем файла.По адресу 1076_8 в тексте авторской программы должен содержаться код 1365_8. После исправления этой неточности программа становится работоспособной.

Программа, повторяю, очень полезна для пользователей, однако у нее есть хороший аналог: программа И.Николаева из г.Москвы (журнал "Информатика и образование". - 1989. - №1). Программа-аналог существенно короче, реализована с использованием подмены того же прерывания EMT 6 и выполняет те же самые функции, что и предлагаемая В.Д.Буртянским. О команде MERGE см. выпуски серии "Вычислительная техника и ее применение", № 9 за 1988 г. и № 3 за 1989 г.

Л.Н.Жариков

Переводы

Шахматный компьютер-чемпион

Как известно, чемпион мира по шахматам Г.Каспаров выиграл две партии, играя против шахматной ЭВМ "Duncom" (Deep Thought), чемпиона среди шахматных компьютеров.

Этот шахматный компьютер -- детище университета Карнеги-Меллона, где для него разработаны специальные микропроцессоры и программное обеспечение. Комментируя ход матча с Каспаровым, специалисты отмечали, что он не стал делать смелые и неожиданные ходы, а, тщательно изучив способности компьютера, в том числе неумение оценивать последствия ключевых ходов и организовывать концентрированную атаку, принял тактику постановки ловушек.

Анализ проведенных компьютером игр показал как его существенные недостатки при начальных ходах, так и сильные стороны, особенно при выборе наилучших вариантов игры в кажущихся безнадежными ситуациях. В отличие от шахматиста, который в невыгодном положении либо сдается, либо делает отчаянные ходы, машина может отдалить поражение, делая наиболее оптимальные ходы и выжидая, когда противник совершит ошибку. Однако Г.Каспаров играл безупречно.

Создатели машины не рассчитывали на победу машины в этом состязании, но отметили, что в одной партии могла быть ничья.

"Scitnce News" (США), том 136, №18, 1989 г.- С.276.

Работникам системы профориентации, подготовки кадров, кадровых служб предприятий и организаций

Программное средство многоцелевого Многостороннего Исследования Личности - МИЛ

В программе МИЛ, разработанной специалистами медицинского объединения "АРС" для ПЭВМ ЕС-1840, IBM PC XT (ОС MS DOS) реализован широко известный в мировой практике психологический тест MMPI.

Испытуемый, ответив на вопросы теста, получает результаты психологического анализа в соответствии с принятыми шкалами, краткую психологическую характеристику, а также график затрат времени при ответе на вопросы теста. Результаты представляются в печатном виде (с принтера) и в графической форме на экране, а также могут храниться в памяти компьютера.

Программа МИЛ найдет применение в системе профориентации, подготовки кадров, в кадровых службах предприятий и организаций при отборе кандидатов на работу с учетом их психологических особенностей. Наиболее важно учесть особенности личности при формировании коллективов, работающих в экстремальных условиях, что обеспечивает улучшение психологического климата, уменьшает вероятность возникновения конфликтных ситуаций.

Программа поможет специалистам - психологам и врачам-психотерапевтам при проведении массовых и индивидуальных обследований, упростив томительную процедуру обработки результатов тестирования.

Программа и документация к ней поставляются на одном ГМД.

Стоимость одной копии:
- для организаций - 1200 руб.;
- для частных лиц - 48 руб.

Отсутствие вирусов гарантируется

Заявки направлять по адресу: 103009, Москва, а/я 346, Медобъединение "АРС".

Для организаций - в форме гарантийного письма на имя председателя медобъединения "АРС".

Частным лицам - перевести через сбербанк 48 руб. на расчетный счет 5461556 в Ленинском отделении жилсоцбанка г. Москвы для Медобъединения "АРС" за программу МИЛ. Письмо направить по адресу "АРСа" с указанием реквизитов перевода (дата, номер сбербанка, номер машины, номер счета) и своего адреса или телефона.

Справки по телефону в Москве: 231-46-28

Аркадий ПЯТКОВ
г. СВЕРДЛОВСК

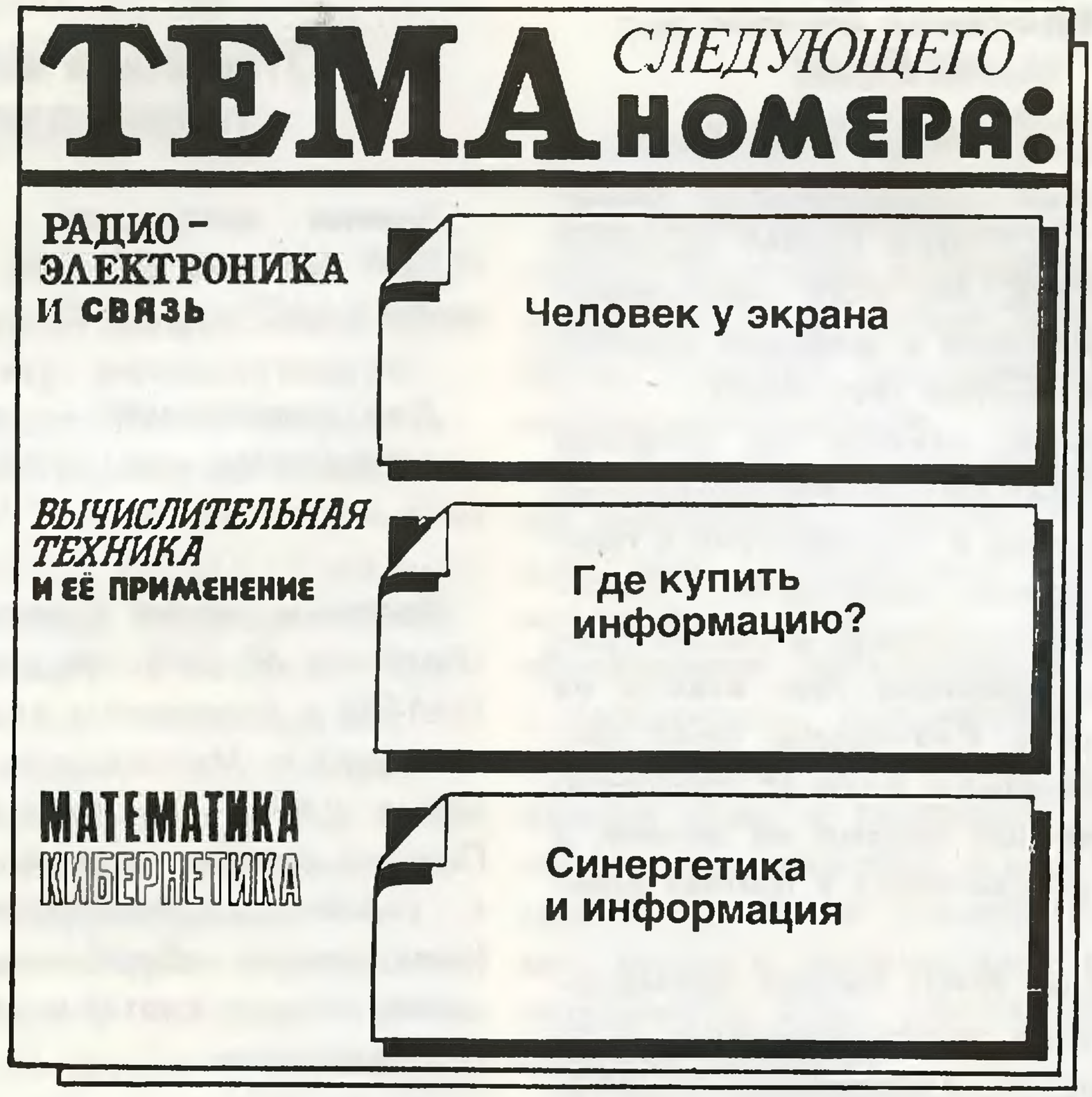
Я 40 **Язык** персональных вычислений. — М.: Знание, 1990. — 48 с. — (Новое в жизни, науке, технике. Сер. «Вычислительная техника и ее применение»; № 4).
ISBN 5-07-001245-2

20 к.

В брошюре дано самое общее представление о языке программирования АПЛ. Этот язык получил широкое распространение за рубежом, однако в СССР мало знают о его мощности и удобстве.

Рассчитана на широкий круг читателей.

2404000000 **ББК 32.97**



Научно-популярное издание

ЯЗЫК ПЕРСОНАЛЬНЫХ ВЫЧИСЛЕНИЙ

Гл. отраслевой редактор *Л. А. Ерлыкин*
Зам. гл. отраслевого редактора *Г. Г. Карвовский*
Редактор *Б. М. Васильев*
Мл. редактор *Н. А. Васильева*
Художник *В. Н. Конюхов*
Худож. редактор *М. А. Гусева*
Техн. редактор *А. М. Красавина*
Корректор *В. И. Гуляева*

ИБ № 10874

Сдано в набор 28.12.89. Подписано к печати 06.03.90. Т00093. Формат бумаги 70×100 1/16. Бумага офс. № 2. Гарнитура гельветика. Печать офсетная. Усл. печ. л. 3,90. Усл. кр.-отт. 8,45. Уч.-изд. л. 4,92. Тираж 73 444 экз. Заказ 868. Цена 20 коп. Издательство «Знание». 101835, ГСП, Москва, Центр, проезд Серова, д. 4. Индекс заказа 904704.
Ордена Трудового Красного Знамени Калининский полиграфический комбинат, Государственного комитета СССР по печати. 170024, г. Калинин, пр. Ленина, 5.

Цена 20 коп.

Индекс 70195

Адрес подписчика:

Сол. 5-27

Подписная научно-популярная серия

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

И ЕЁ ПРИМЕНЕНИЕ

В будущем окажется возможным. . . создать такие кибернетические управляющие системы, которые позволят человеку управлять удаленными от него машинами посредством одних только мысленных команд.

Сиборг

Машина становится не только помощником, но и конкурентом в интеллектуальной деятельности человека

И.М. Лифшиц

Наш адрес:
СССР,
Москва,
Центр,
проезд
Серова, 4